UNIVERSITA' DEGLI STUDI
DI MILANO
DIPARTIMENTO DI SCIENZE
DELL'INFORMAZIONE

Honeywell Bull
CENTRO STUDI
INFORMATICA
E AUTOMAZIONE

NOTE DI SOFTWARE

ARCHIVIO
STORICO
ASSOCIAZIONE
POZZO DI MIELE
FPDM-251
RNSW 128

ANNO 1988
41
OTTOBRE

Note di software è una pubblicazione trimestrale, edita a cura del Dipartimento di Scienze dell' Informazione dell'Università di Milano e dal Centro Studi Informatica e Automazione della Honeywell Bull Italia.

Note di software intende costruire uno strumento per la circolazione di informazioni, idee, metologie ed esperienze nell'area delle tecnologie e della produzione del software.

La collaborazione a **Note di software** è libera. Chi desiderasse pubblicare un contributo è pregato di prendere contatto con il Direttore responsabile o con il Comitato di Redazione. In particolare si sollecitano contributi nella forma di monografie, da inviare in triplice copia, e che non dovrebbero superare le cinquemila parole.
I lavori ricevuti per la pubblicazione verranno revisionati dai membri del Comitato di Revisione Scientifica, che possono avvalersi della collaborazione di specialisti del settore. I lavori pubblicati riflettono comunque le opinioni dei loro autori.

Note di software viene distribuito ad Aziende, Enti e specialisti del settore che ne facciano richiesta alla Redazione.

Direttore Responsabile: Franco Filippazzi
Honeywell Bull Italia S.P.A., Centro Studi Informatica e Automazione, via Vida 11 - 20127 Milano

Comitato di Redazione: Stefania Bandini, Francesco Gardin, Roberto Polillo
Università di Milano Dipartimento di Scienzee dell'Informazione
via Moretto da Brescia, 9-20133 Milano tel. (02)2772233

Comitato di Revisione Scientifica: Alberto Bertoni, Gianluigi Castelli, Giovanni Degli Antoni, Giorgio De Michelis, Mario Italiani, Gaetano Aurelio Lanzarone, Giancarlo Martella, Marco Maiocchi, Giancarlo Mauri.

Supporto Tecnico-gestionale e Distribuzione: Domenico Asinelli
Honeywell Bull, via Tazzoli 6 - 20154 Milano tel. (02) 67792960

Autorizzazione del Tribunale di
Milano n. 87 del 23 febbraio 1977

FPDM - 251

UNIVERSITA' DEGLI STUDI
DI MILANO
DIPARTIMENTO DI SCIENZE
DELL'INFORMAZIONE

Honeywell Bull
CENTRO STUDI
INFORMATICA
E AUTOMAZIONE

QUESTO NUMERO 1

- IL COMPUTER E IL REALE
  Giovanni degli Antoni 3

- ARTIFICIAL INTELLIGENCE AND COMPLEX DYNAMICS
  Luc Steels 6

- MODELLI QUALITATIVI PER IL SUPPORTO DECISIONALE
  Mauro Cadei, Francesco Gardin 27

- PROGRAMMAZIONE LOGICA E RAPPRESENTAZIONE DI CONOSCENZA STRUTTURATA: IL CASO DEI FRAME IN PROLG
  S. Bandini, G.A. Lanzarone, A. Messina 44

- LINGUAGGI OBJECT- ORIENTED: IMPLEMENTAZIONI DI UN SISTEMA A FLAVORS IN LISP
  Danilo Dabben. 53

- SOME RESULTS IN COMPLEXITY OF MATCHING, UNIFICATION, REWRITING PROBLEMS
  Maria Paola Bonacina 58

AVVENIMENTI

- OPTICAL INFORMATION SYSTEMS 1988
  Giancarlo Dalto, Renzo Magni 75

RECENSIONI

- a cura di Mario Italiani e di Ugo Solitro 77

NOTE DI SOFTWARE

ANNO 1988
41
OTTOBRE

QUESTO NUMERO

Con questo numero, la rivista presenta un avvicendamento del direttore responsabile. A causa di propri impegni professionali Paolo Ghelfi lascia tale carica - da lui ricoperta fin dalla nascita della rivista- a Franco Filippazzi.

Nel ringraziare Paolo Ghelfi per il fattivo apporto dato in tutto questo periodo, la Redazione dà il benvenuto al nuovo direttore responsabile. Franco Filippazzi ha vissuto l'informatica italiana sin dalla sua fase pionieristica e potrà portare alla rivista il contributo della sua esperienza.

Cogliamo questa occasione per ricordare come Note di Software abbia ormai superato i dieci anni di vita. In questo periodo si è venuto consolidando il ruolo che la Rivista si era assegnato, quello cioé di costruire un veicolo qualificato per la diffusione nell'ambiente accademico ed aziendale italiano di idee, progetti e risultati generati, in particolare, nel nostro Paese. Il consenso sin qui raccolto ci è di sprone a perseguire con sempre maggiore incisività tale obbiettivo.

ooo

Questo numero si apre con uno stimolante contributo di G. Degli Antoni sul computer come "realtà artificiale". Questo strumento, in effetti, sintetizza la percezione, dà consistenza all'immaginato e consente esplorazioni altrimenti impossibili.
L'articolo successivo di L. Steels individua il nuovo paradigma emergente nella ricerca in Intelligenza Artificiale, caratterizzato dal fatto di considerare un'operazione di calcolo come un sistema di calcolo complesso.
Nel terzo articolo, M. Cadei e F. Gardin descrivono l'utilizzo di tecniche di ragionamento qualitativo quale parte integrante di un sistema di supporto alle decisioni.
S. Bandini, G. A. Lanzarone e A. Messina presentano un lavoro sulla rappresentazione di frame in Prolog e illustrano un sistema basato su tale approccio.
D. Dabben descrive uno sviluppo connesso ai linguaggi object-oriented, ossia l'implementazione di un sistema a Flavors in Lisp.
Nell'ultimo articolo, M. P. Bonacina presenta una rassegna sulla complessità compiutazionale di tre problemi di base nell'ambito della dimostrazione automatica dei teorenni.

LA REDAZIONE

IL COMPUTER, IL REALE, L'ARTIFICIALE

Giovanni Degli Antoni
Dipartimento di Scienze dell'Informazione
Università di Milano

IL COMPUTER ED IL REALE

Il computer è una parte del reale come qualsiasi altro strumento di cui ci si serve nella vita quotidiana. E parte del reale è l'impiego del computer: scrivere testi, disegnare, effettuare calcoli, costruire modelli. E reale è la attività che è necessaria per quell'impiego.
Non tutti i testi, disegni, calcoli o modelli tuttavia si riferiscono al reale. Sono spesso il risultato di idee costruite dalla immaginazione. Ma anche quando testi, disegni, calcoli o modelli si riferiscono al reale, diventano altro. Sempre trasportano immaginazioni costruite attraverso osservazioni, astrazioni, analogie o invenzioni. Ma testi, disegni, calcoli o modelli non richiedono un computer per essere costruiti. Però senza un computer tali entità rimangono puri riferimenti agli atti mentali che li hanno costruiti o ad altri che potranno ricostruirli.
Con il computer testi, disegni, calcoli e modelli sono associabili. Ne nascono computers che si comportano come altro, cioè come non computers o altri computers.
Così sono stati realizzati computers che sibilano comportandosi come una locomotiva di cui simulano il comportamento.
Computers che assomigliano ad edifici in fase di progettazione.
Computers che si comportano come batteri nella lotta per la loro sopravvivenza.
Computers che si comportano come missili antimissile nella loro estrema volontà di colpire.Computers che si comportano come una reazione chimica durante il suo svolgimento.
Computers che si comportano come robot durante una missione...
Tutti questi computers sembrano piccoli universi che contengono una realtà.

Ma questa realtà non è esattamente naturale. Senza gesti umani e senza computers attivi quella realtà non esisterebbe.
Quella realtà, che forse potremmo chiamare realtà artificiale è creata dall'uomo attraverso i computers.
Con quei computers l'uomo forza la natura: la osserva, la analizza, ne fa astrazione, la sintetizza nei computers: fa sì che questi computers si comportino come il reale di cui si intende prevedere il comportamento sulla base delle leggi osservate.
E, miracolo, descrive il tutto con la parte. Con qualche rischio. Ma lo fa. Il computer è diventato lo specchio in cui il reale rivive una sua nuova realtà al di fuori dello spazio e del tempo: la realtà artificiale.

IL COMPUTER COME REALTA' ARTIFICIALE

La possibilità di osservare il computer come realtà artificiale mette il progettista in una situazione particolarmente creativa. Chi legge il termine "realtà artificiale" si trova di fronte ad una sorta di metafora che lo porta dalle sue conoscenze ad un nuovo mondo.
Gli elementi di questo nuovo mondo gli sono in parte conosciuti. Infatti della realtà artificiale dovrà far parte nella sua massima generalità la immagine del mondo reale che conosce più o meno bene. Ma nella realtà artificiale dovrà entrare gradualmente anche il mondo delle idee con tutti i suoi meccanismi per produrre nuove idee ed altri

mondi.
Fino a che punto ciò sia possibile è ancora presto per stabilirlo. Per ora è conveniente osservare come lentamente il processo sia in atto. Pazientemente l'informatica sta costruendo universi artificiali sempre più articolati.
Vediamo storicamente cosa è avvenuto. Prima si è visto l'elaboratore come strumento per riprodurre operazioni essenzialmente numeriche. Poi si è visto che l'elaboratore elettronico può fare operazioni su oggetti rappresentati simbolicamente. Si è cominciato a rappresentare enti geometrici. Gli enti geometrici sono entrati in movimento. Un po' dell'astrazione geometrica e cinematica dell'universo era entrato nel mondo. Ora c' è chi a quella astrazione sta associando forme.
L'idea del CAD è di modellare oggetti: hanno una forma, un colore, eventualmente gli oggetti hanno attributi o costituenti (colore e proprietà di plasticità ad esempio) e possono muoversi in un loro spazio.
Lentamente con il ridursi del costo del calcolo anche la dinamica entra nel computer: gli oggetti rappresentati si comporteranno come entità dotati di massa nel senso fisico del termine: forse ciò non farà guadagnare comprensione su ciò che sia realtà dinamica. Ma permetterà di disporre dei computers in cui, ad esempio, gli oggetti fragili si rompono. Basterà disegnare l'oggetto (un vaso per esempio). Basterà attribuirgli un materiale fragile. L'oggetto dovrà essere messo in un campo di forze. Dovrà incontrare un ostacolo. Si romperà in parti che il computer dovrà descrivere. Ma come abbiamo detto non solo l'astrazione del mondo fisico sta entrando nella realtà artificiale.
Interi universi di immagini vengono in questi ultimi tempi immessi nei computers. Sono immagini del mondo reale o immagini artificiali (immaginate), fisse o in movimento. E le sequenze non sono controllabili o lo sono per poter scorrere queste immagini come se si camminasse nella realtà artificiale.
Si rappresentano verità in forma logica o in forma analogica.
Ma non si ferma certo alle immagini.
Si manipolano verità con inferenze logiche ed analogiche.
La forma logica si porta nella realtà artificiale ciò che i filosofi hanno sempre studiato e cercato di rappresentare con le loro assiomatizzazioni.
E' ancora poco. E si sa fare ancora male. Ma se le assiomatizzazioni dei filosofi catturano idee, nella realtà artificiale ci sono idee artificiali.
La forma analogica delle verità è rappresentata direttamente in forme, oggetti, movimenti, entità.
La rappresentazione analogica della verità è di recente acquisizione ed ancora non ben capita, ma certamente corrisponde ad una caratteristica della comprensione umana del reale.

Qui molte obiezioni saranno state fatte: ma insomma, l'informatica costruisce una realtà artificiale diversa dal soggetto osservatore o costruisce una realtà in cui c'è anche l'osservatore?
La risposta ovvia (entrambe) non merita per ora molta attenzione: è meglio dimenticare quel problema che procura solo grattacapi. Conviene, come la storia ha insegnato, considerare che tutto ciò che si osserva è osservabile e quindi oggetto del reale. Lo metteremo nel mondo delle idee, ma il soggetto non c'entra. D'altra parte non ne abbiamo per ora più bisogno dal momento che la realtà viene costruita vivente nell'universo di qualche computer. Vedremo che il soggetto farà la sua ricomparsa, intero, non più preoccupato di un problema che lo ha sempre turbato.

REALTA' ARTIFICIALE E PERCEZIONE

Fin qui, nulla di nuovo. Certo, il vedere la crescita dell'informatica come trasformazione del reale e dell'immaginato in una nuova realtà aiuta a capire la dinamica della crescita delle tecnologie dell'informazione e permette di immaginare facilmente nuove tecnologie informatiche.
Certo è più facile realizzare che il cammino della integrazione con la informatica è appena iniziato. Ancora: è sempre più chiaro che la informatica si contraddistingue per sapere integrare il risultati delle proprie ed altrui analisi (cioè di tutte le altre scienze). Ma si può procedere ancora un po'. Infatti l'informatica ha anche insegnato ad osservare via computer il reale direttamente: a misurarlo, a rappresentarlo ed in qualche caso ad effettuare delle astrazioni sul reale.
Ma dunque l'informatica può inserire i risultati di quelle osservazioni/misura, osservazioni/rappresentazioni, osservazioni/astrazioni direttamente nei computer o sul video nel contesto appropriato di una realtà artificiale che ricostruisce il mondo a cui le dette osservazioni si riferiscono. Finalmente il risultato di una osservazione non è solo misura, non è solo rappresentazione, non è solo astrazione.
Il risultato è percezione del reale nella sua ricostruzione.
E' percezione della parte fisica del reale: è percezione della parte ideata e che diventa realtà artificiale in cui sono inserite le misure, le rappresentazioni, le astrazioni.
Prima quel mondo non c'era. Ora esiste, come tutti gli altri mondi. Abbiamo finalmente reso realtà (artificiale) la percezione non più disintegrata dalla immaginazione.

ENTRARE NELLA REALTA' ARTIFICIALE

Il problema si pone immediatamente.
Se la realtà artificiale si può osservare allora si può entrare nella realtà artificiale.

Se si può entrare nella realtà artificiale allora si può disporre di tutto il reale reso reale/artificiale vicino a noi, ovunque noi ci troviamo.

Un esempio potrà chiarire l'idea.

Immaginiamo dunque un Ministero artificiale disegnato sul video di un computer.
L'UTENTE di quel computer vorrà entrare in quel Ministero per certe sue pratiche. Dovrà esporre quesiti su certe questioni.
All'UTENTE il Ministero apparirà come un edificio con una porta di ingresso.
Fra le opzioni che il computer gli offrirà l'UTENTE sceglierà quella di entrare. Indicherà tale volontà con un gesto opportuno impiegando quelli che vengono definiti gli elementi della interfaccia uomo-computer, che diventano elementi con cui l'UTENTE interagisce con la realtà artificiale.
Ad esempio toccherà la porta di ingresso con un indicatore mobile che si sposta seguendo il movimento di un dispositivo chiamato mouse che l'UTENTE stesso sposta su un piano.
La porta si apre e l'UTENTE vede ora la portineria cui dovrà lasciare un suo documento di identificazione che permetta alla portineria di identificarlo.
L'UTENTE ora potrà proseguire il suo percorso lungo corridoi.
Sui lati di questi vedrà nomi di uffici. Su ogni ufficio vedrà il nome dell'impiegato. Finalmente vede l'ufficio che gli interessa. Legge il nome dell'impiegato. Si informa sull'impiegato stesso avendo accesso all'archivio che contiene i profili professionali degli impiegati.
Gli basterà premere un bottone di un apposito video che è lì, a fianco del nome dell'impiegato.
Su quel video compaiono molti bottoni.
Ciascun bottone permette di leggere, se premuto, altre informazioni: funzioni e ruoli dell'ufficio, orari di accesso (anche nella realtà artificiale un ciclo sonno-veglia è utile per poter riordinare dati), vacanze in cui il Ministero è chiuso, curriculum vitae degli impiegati, costo dei servizi, eccetera.
Ora il nostro UTENTE entra nell'ufficio.
In questo osserva i volumi delle pratiche ben ordinate.
Vede l'impiegato al lavoro. Sa che l'impiegato in assenza di pubblico si adopera affinché il suo ufficio sia il più informato possibile, il più disponibile possibile. L'UTENTE non sa porre bene il quesito. Per cui l'impiegato a voce e per iscritto gli presenta i quesiti più frequenti che in quell'ufficio vengono risolti. Glieli presenta a voce e per iscritto. Sembra un po' di giocare: "Vuole sapere la disponibilità di finanziamenti sulla legge 64 o come si presentano i progetti?". L'UTENTE risponde.
L'impiegato non sa da solo risolvere i dubbi.
Ha bisogno dell'aiuto di un impiegato vero, quello del Ministero Reale a cui la realtà artificiale è connessa. Lo chiama per videotelefono.
Compare l'impiegato. Inizia una conversazione nell'ufficio del Ministero Artificiale con l'impiegato del Ministero reale. Il quesito viene risolto.
L'Impiegato Artificiale ha seguito con cura la conversazione. D'ora innanzi saprà risolvere anche quel quesito.
L'UTENTE ora può uscire e continuare il suo viaggio nella realtà. Entrerà questa volta nei suoi uffici. Osserverà lo stato del progetto dei Laboratori di Ricerca che dovrà progettare per il mese successivo. Entra nel progetto. Dall'interno osserva con giudizio critico forme e colori. Li modifica, provando e riprovando.
In poche decine di minuti ha potuto vedere non dal vero, ma quasi, il suo progetto in varie forme: ha così acquisito in poche ore quella esperienza che solo dopo molti anni un architetto può acquisire.
Gli esempi possono facilmente continuare.
Il lettore ha ben compreso, a questo punto, che la realtà artificiale è una matassa creativa facile da svolgere che trasforma il computer in uno strumento per ripensare.
E' una sorta di generatore di metafore o di storie che diventano particolarmente interessanti se applicate alle applicazioni dei computers nel mondo di oggi.
E non è difficile accorgersi che queste realtà artificiali già esistono oggi: simulatori di volo, giochi di vario genere, simulatori di organizzazioni, computers per provare trucchi, computers per indossare abiti, eccetera.
Realizzare realtà artificiali è come dar corpo alle immaginazioni: i limiti di questo modo di procedere sono ancora tutti da scoprire, da indagare e da capire. Ma la strada è tracciata. E nella realtà artificiale, lentamente, anche l'uomo farà la sua comparsa.
Potremo entrare nella sua struttura fisica, nella sua struttura mentale. Potremo modificarlo. O potremo capire come funziona. Lo specchio sarà completo, ma non sapremo più se lo specchio è il reale o l'artificiale.

CONCLUSIONE

Questa nota ha obiettivi definibili dal lettore.
E' una semplice e sintetica riesplorazione dei mondi artificiali che la informatica e la telematica rendono possibili.
Il lettore potrà ritrovare nell'immaginazione che gli viene proposta anche una parte del suo lavoro. Se lo avrà ritrovato non si limiti a dire: lo sapevo già! Cerchi piuttosto davanti a quella verità trovata sul suo percorso le strade che lo conducono a luoghi che non conosceva. Se avrà avuto pazienza li troverà.

ARTIFICIAL INTELLIGENCE AND COMPLEX DYNAMICS (*)

Luc Steels
VUB AI Lab - Bruxelles

RIASSUNTO

Questo articolo individua un nuovo paradigma emergente nella ricerca in Intelligenza Artificiale caratterizzato dal fatto di considerare un'operazione di calcolo come un sistema dinamico complesso.
Dopo una breve introduzione ad alcuni concetti fondamentali della dinamica complessa, verranno presentate alcune classi di sistemi di calcolo dinamico. Questi comprendono macchine cellulari, sistemi connessionistici, reti a rilassamento, algoritmi genetici e sistemi di tipo reazione-diffusione.
L'articolo si conclude con una descrizione di come questo nuovo paradigma affronta in modo assolutamente innovativo alcuni problemi fondamentali dell'Intelligenza Artificiale, quali ad esempio rappresentazione, ricerca, controllo ed apprendimento.

ABSTRACT

The paper identifies a newly emerging paradigm in Artificial Intelligence research characterised by viewing computation as a complex dynamical system. After a short introduction in some fundamental concepts of complex dynamics, several classes of dynamical computation systems are reviewed. They include cellular automata, connectionist systems, relaxation networks, genetic algorithms and reaction-diffusion systems. The paper concludes with an outline of how the new paradigm is approaching in a novel way key questions in artificial intelligence, such as representation, search, control and learning.

1. INTRODUCTION

Since its beginning 30 years ago, Artificial Intelligence has made impressive progress based on adopting a symbolic paradigm for looking at cognitive phenomena. Symbols are viewed as primary units. They can be assembled to form structures which are used to represent parts of the world, intermediate results in problem solving; perceptual input, associative memories, inference rules, etc. Interpreters create structures and operate over them under the control of programs that are themselves represented explicitly.

Some important milestones in the elaboration of the symbolic paradigm were the definition of a symbolic level of computation [41], the discovery of the state space search model of problem solving and subsequently of the need for strategic and heuristic knowledge [49], the translation of linguistic formalisms into computational models [73], the mechanisation of logical inference [39], and the realisation that learning is a search process [71]. Applications such as expert systems [23] or natural language interfaces are directly based on these insights.

The symbolic paradigm rests on concepts and theories elaborated mostly in the humanities (philosophy, logic, linguistics and psychology). It is remarkable that it was possible to transform these concepts into structures that are implementable in computational media. Neither were they developed for this purpose, nor was it is easy to bridge the gap between mentalistic notions, such as the notion of a goal, and its computational counterparts.

(*)Paper presented at the IFIP workshop on concepts and tools for knowledge-based systems. Mount Fuji, Japan. November 1987.

At the moment we see the beginnings of a new paradigm for approaching cognition that could very well dominate the next 30 years of artificial intelligence research. This paradigm rests on concepts and theories from the natural sciences (physics, chemistry and biology), and particularly from scientific theories of complex phenomena such as statistical mechanics, physical chemistry, developmental biology, or genetics. The phenomena studied in these fields are complex in the sense that they involve massively parallel interaction of a large number of elements, e.g. billions of interacting molecules. Because the key idea is to view computation as a complex dynamical system, I call this new paradigm the dynamics paradigm.

The push for the new paradigm comes from two sources: technological and scientific.

1.1 Technology Motivation

It is now possible to construct computer systems that consist of a large number of processors (at present on the order of 60,000 [31] and this number will increase in the future. Consequently it starts to become feasible to synthesise complex phenomena directly with digital hardware. There are also opportunities for constructing systems based on analog VLSI circuitry [43], optics [4], or even molecular components [10], that may raise the level of parallelism even more and bring computers closer to natural complex dynamical systems. Programming these massively parallel 6th generation computers requires a completely different approach than the standard hierarchical paradigm used for programming serial or small-scale parallel machines (i.e. machines with hundreds of complex nodes such as the Intel Hypercube or the BBN Butterfly for which languages like Concurrent Prolog [59] or Concurrent LISP might be appropriate). Specifically, a simple parallelisation of current concepts leads to increasing overheads for synchronisation, scheduling, protection of shared resources, etc. as the amount of parallelism increases.
An entirely different approach is therefore needed with only local interaction and no central control structure. Artificial Intelligence is obviously affected, if only because the brain is manifestly also a complex dynamical system, and the new programming concepts needed for massively parallel machines may throw new light on the functioning of the brain itself.

1.2 Scientific Motivation

Despite the enormous achievements of the symbolic paradigm, there are a large number of issues in Artificial Intelligence that remain unresolved. Consider the following non-exhaustive list:

a. Flexibility and robustness: No AI system matches the flexibility of the human brain, which tolerates deviations in input (e.g. in pattern recognition), adapts itself easily to slightly different circumstances, shows graceful performance degradation in problem solving, etc. Moreover AI systems are not robust, in the sense of being fault-tolerant. Every unit, whether it is a concept, rule, or some other construct, is there to serve a particular function and if the unit malfunctions the whole system fails. In contrast, the brain is able to keep functioning even if certain units disappear or if parts are damaged.

b. Problems in decomposition: The principle of functional decomposition which underlies all hierarchical systems, including current computer programs, assumes that a problem can be split up in different subproblems which can be solved independently. A solution for the total system is then obtained by connecting together in an appropriate way the subsystems that solve each subproblem. This approach has a great number of advantages, for example modularity. But many problems in AI are not decomposable this way. For example, many interpretation tasks (natural language understanding, vision, measurement interpretation, etc.) involve ambiguous data. The data itself does not contain enough information to uniquely interpret it so that knowledge of the situation or general world knowledge must be brought to bear. This leads to a chicken and egg situation: we need to partially interpret the data but to interpret the data we need a partial interpretation. No straight division into subproblems is possible. Similar difficulties arise in planning (e.g. the conjunctive subgoal problem [65] or inference.

c. Explicit construction: Most AI programs today are explicitly constructed by hand. This is feasible for small-scale problems, but the complexity barriers are soon reached when realistic problems are tackled. Even for small domains, the difficulties of knowledge extraction, as experienced in the construction of expert systems, are very severe. When parallelism is involved, the problem becomes even more serious because issues of synchronisation, protection of shared resources, scheduling, etc. have to be taken into account. It is true that recently there has been substantial progress in the area of machine learning (see [45] for a representative sample). But the distance to systems that have acquired a complete domain of expertise is still enormous. Moreover learning programs are themselves becoming progressively more complex raising the question how they are formed in the first place.

The current approach to these fundamental problems is to

add more and more knowledge to AI systems: knowledge on what to do when the input is not exactly as required, knowledge on flexible control, knowledge on synchronisation and scheduling of parallel reasoning sequences, etc. This exagerates the problems of knowledge extraction and engineering complexity. Sometimes complex bureaucratic systems are proposed even for relatively simple tasks. These systems recurse to a tower of meta-levels in an attempt to deal with issues of flexibility, control of reasoning, etc. [41]. Although knowledge will remain the key ingredient of any AI system, maybe it is time to look for a whole new set of tools and concepts for building knowledge based systems. There are indications that the dynamics paradigm offers genuine alternatives on all these issues and may therefore be the source for this new set of tools. At the same time one should not exagerate either. The symbolic paradigm remains the only viable option to build knowledge-based systems in the near future. The concepts and tools proposed by the dynamics paradigm should at this point be seen as complementary.

1.3 Overview

In this paper, the state of the art in the exploration of the dynamics paradigm is reviewed. The field is in great flux and developments take place at a rapid pace. Many of the researchers exploring the dynamics paradigm are new to the field of AI but have a long standing experience with dynamical systems. The literature is spread over a large range of scientific journals, many of them not traditionally read by the AI community. There are as yet few conferences that focus exclusively on the new paradigm. Constructing a review of the state of the art is therefore a difficult enterprise.

Given that the essence of the new paradigm is the notion of complex dynamical system, the first part of the paper contains a brief elementary introduction to complex dynamical systems. The second part focuses on how dynamical systems can be used for computation. The third part gives an overview of the various classes of systems that have been studied so far in direct connection with Artificial Intelligence. The final part discusses briefly what the new paradigm may offer concerning some of the key questions in the field.

2. INTRODUCTION TO COMPLEX DYNAMICAL SYSTEMS

2.1 What Is A Complex Dynamical Systems

A dynamical system is a system in which a number of properties evolve in parallel over time. The classical and simplest example is a pendulum. (Fig 1.a.) Its dynamics can be described in terms of two properties: the angle of elevation θ and the rate of rotation $\dot{\theta} = d\theta/dt$. Each of these θ follows a specific trajectory made up by the values they take at each point in time. Another example are the gas molecules in a container (Fig 1.b.) Each of these molecules constantly changes position in time.

Dynamical systems abound in nature. Examples include an object falling to earth, evolving animal populations, the growth of an organism, planets moving around the sun, evolution in the stock market, or traffic flow. Even standard computer programs can be viewed as dynamical systems. For example, the program counter, which indicates control flow, continuously changes in time as the computation proceeds.

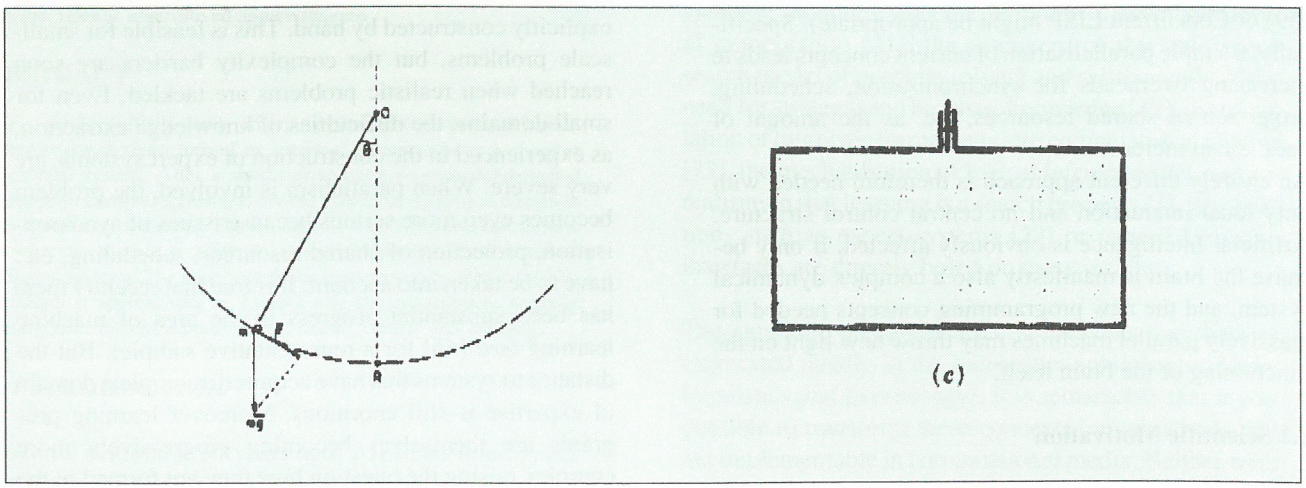


Fig. 1 Examples of dynamical systems: a. pendulum. b. gas molecules.

A distinction can be made between simple and complex dynamical systems, based on the number of properties undergoing change. In the case of the pendulum, the falling object, or the computer program, only very few properties keep changing. These properties can therefore be described and analysed explicitly. In the case of the gas container, there is an extremely large number of properties (e.g. a drop of water contains already billions of molecules). It is therefore no longer feasible to follow the course of each individual property, only statistical or macroscopic statements can be made.

Even more important is the following distinctive feature: Because of the many interactions between the evolving properties in a complex dynamical system, a new set of behaviors emerges. These behaviors are not the consequence of a single element in the system, but a side effect of the collective behavior of all units. Complex dynamical systems therefore differ from simple dynamical systems because they show emergent properties.

These emergent properties are often very remarkable and usually occur suddenly at phase transitions. For example, there are certain reactions (such as the Belousov-Zhabotinsky reaction) that after a while lead to an oscillating behavior (i.e. a chemical clock) or to the formation of patterns (Fig 2.). It is as if billions of molecules cooperate on cue, without a supervising global control mechanism, or global communication. [2]

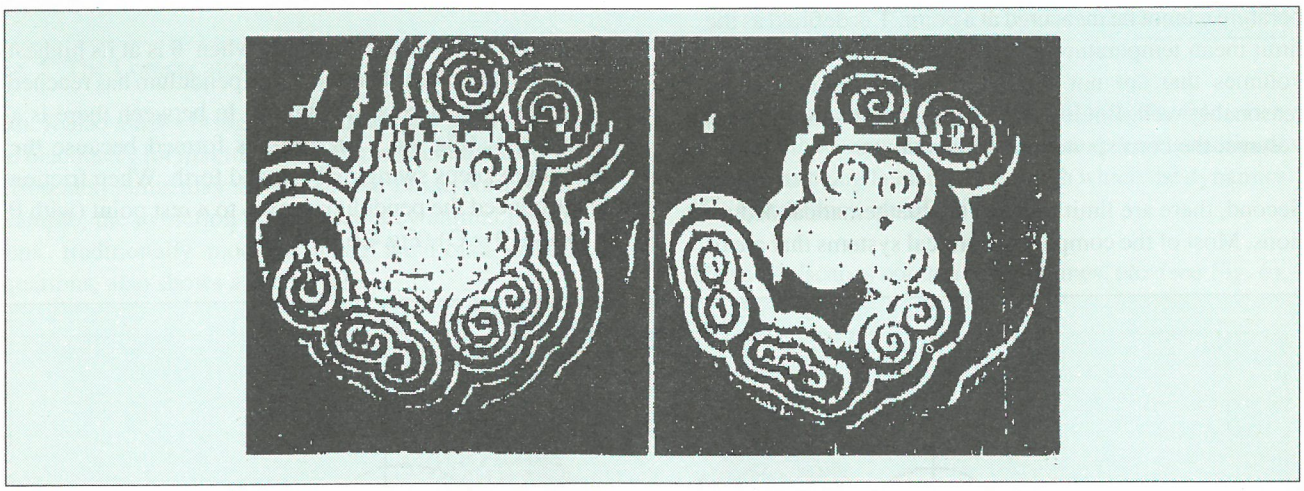


Fig. 2 Example of emergent properties in Belousov-Zhabotinsky reaction.

Another example is the formation of a path by an ant society [16]. The changing position of the ants form a complex dynamical system. Each ant in principle moves around more or less randomly but occasionally the ants spontaneously organise themselves to establish a path between a food source and the nest. The path shows properties of flexibility and robustness that were discussed earlier as problematic for hierarchical symbolic systems. Removing an ant makes little difference to the whole path, so the system is robust. Obstacles can be put on the path and the ants will find their way around it, showing flexibility to adapt to new circumstances.

Yet another example is genetic evolution through natural selection [15]. The dynamical system is made up of genes which are copied, mutated and crossed over. Gene populations evolve over time controlled by mechanisms enforcing fluctuations that lead to better adapted organisms. This example demonstrates that intricate and complex structures wich surpass human engineered devices in complexity can be obtained in dynamical systems without any explicit construction by an external agent or a central supervisor controlling the parallel and fully distributed activity. The immune system appears to work on the basis of the same selectionist principles [19].

2.2 Describing Complex Dynamical Systems

There are three ways to describe dynamical systems: differential equations, geometrical descriptions, and computational models.

2.2.1 Differential Equations

A simple dynamical systems can easily be described by a set of differential equations which define the rate of change of the various properties in the system. Such a description is particularly useful because many mathe-

matical deductions can be made based on it. One of these is integration which yields the evolution in time of various quantities. For example, given a simple chemical reaction (3.a), the rate of change of A is described by the differen

$$A \xrightarrow{k_1} B \qquad (a)$$

$$\frac{dA}{dt} = -k_1 A \qquad (b)$$

$$A = A_0 e^{-k_1 t} \qquad (c)$$

Fig. 3 Different descriptions of the chemical reaction.

tial equation (Fig 3.b.) from which a function (Fig 3.c.) for computing the evolution of A in time can be deduced by integration.

There are two caveats however. First of all, these differential equations, particularly in the case of complex dynamical systems, rest on an approximate model of reality. Toffoli [66] discusses in this context the heat equation which involves the temperature T. Since temperature cannot be measured at a point, T is defined as the limit mean temperature as the volume goes to zero. For volumes that are not too small this definition works reasonably well. But if we already start with a very small volume, the correspondence breaks down.

Second, there are limitations to the mathematical deductions. Most of the complex dynamical systems that are of interest in the present context require nonlinear partial differential equations of higher order. Only a few excep

tional cases can be solved analytically. This makes the mathematical machinery which is so useful for studying simple dynamical systems almost useless!

2.2.2 Geometrical Dynamicis

An alternative mode of description pioneered by Poincarre is known as geometrical dynamics. Concepts from geometrical dynamics will be used in the rest of the paper so at least a few words of explanation are needed. (The interested reader may consult [1] for a more extensive introduction.)

The basic idea is to represent the evolution of the quantities making up the dynamical system in terms of trajectories through a phase space. Each point on this trajectory describes a configuration of the complete system (more precisely those aspects that are considered relevant in the analysis of its dynamics). The time dimension is assumed to be the third dimension but not explicitly represented. Instead points are mapped into a phase plane. The resulting graph is called the phase portrait. For example, the two quantities R and A describing the pendulum could be represented in the phase portrait of Fig. 4.

In the case of an ideal pendulum, when θ is at its highest value, θ will be O. When $\theta = 0$ (the pendulum has reached the axis), θ will be at its highest. In between there is a continuum. (Fig 4.a.) An ellips is formed because the pendulum keeps swinging back and forth. When friction is introduced the pendulum moves to a rest point (with $\theta = O$ and $\dot{\theta} = O$) (Fig 4.b.).

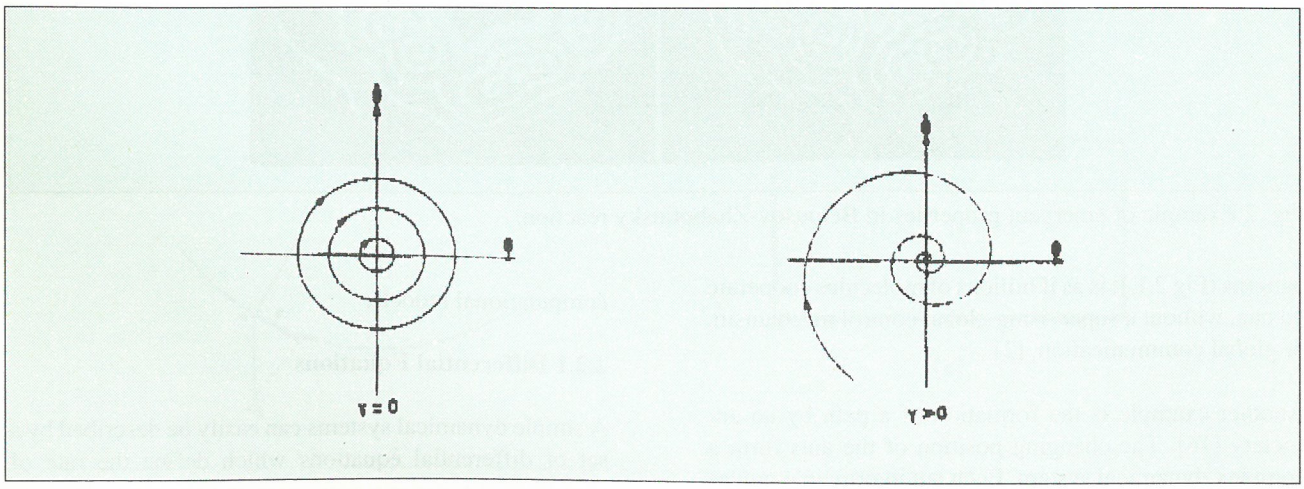


Fig. 4 a. Phase portrait of ideal pendulum: b. Phase portrait of pendulum with friction.

This example illustrates that the trajectories in a phase portrait evolve towards specific regions. These regions are called attractors. When the region is a specific point, they are called point attractors. A pendulum with friction has a point attractor. From whatever initial position, the system evolves to this point. Another example of an attractor is the limit cycle which is formed when the same configuration keeps reoccurring occasionally. Thus the ideal pendulum shows a limit cycle.

Note that the geometrical representation yields an intuitive visualisation of the evolution of the dynamical sys

Systems may have several point attractors. Depending on the initial condition evolution may go towards one point attractor or another one. There are even strange attractors with complex but recurring trajectories. An example of this is the Lorenz attractor displayed in Fig 5. These strange attractors underly chaotic dynamical systems [17].

In the geometrical dynamics approach, the study of complex dynamical systems amounts to a description of the attractors that may occur in the state space, and particularly of the transition between different attractors,

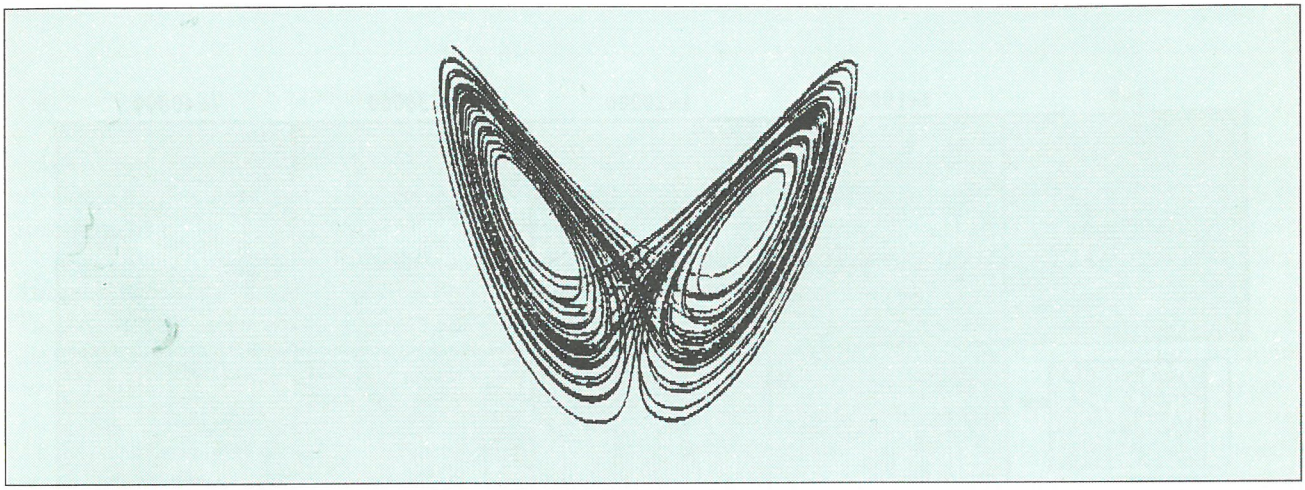


Fig. 5 The Lorentz attractor.

tem. It also allows bringing in geometrical and topological machinery for making mathematical deductions. Very different systems may show a similar phase portrait. For example, the evolution of predator-prey animal populations, traditionally modeled using the Lotka-Volterra equations, also shows a limit cycle.

e.g. due to phase transitions and bifurcations. One often imagines this complex of attractors and their transitional surfaces as being landscapes through which the dynamics of a system may travel.

These landscapes contain hills, ravines, etc. (see Fig. 6).

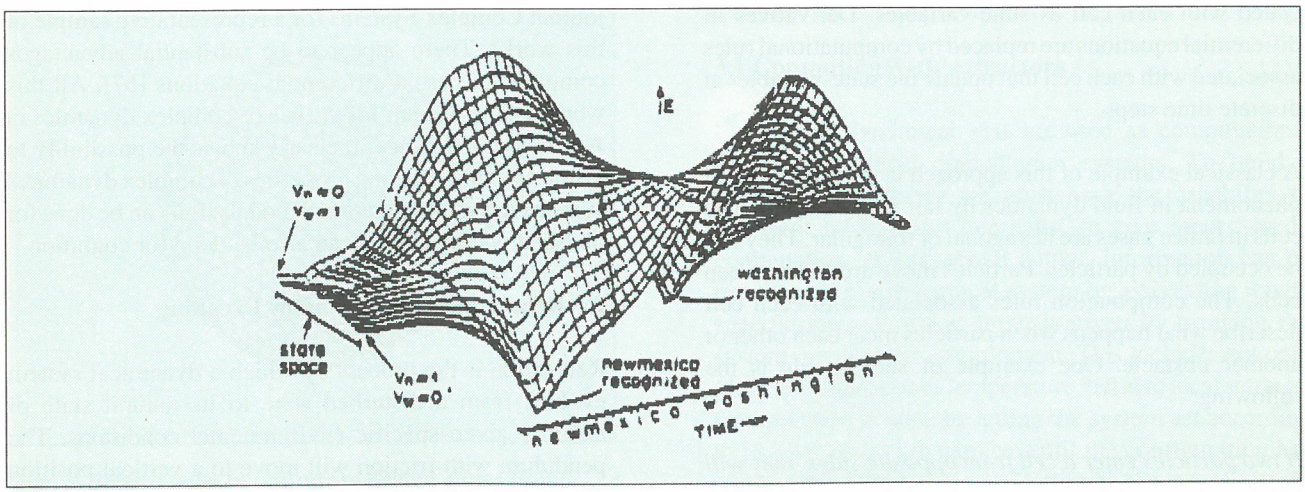


Fig. 6 Attractor landscape associated with Hopfield net (see section 2.3.1). Two trajectories are shown.

All sorts of techniques have been developed to aid in this analysis and to relate phase portraits with differential equations. (See [5] for a wide variety of examples.)

2.2.3 COMPUTATIONAL DESCRIPTIONS

The computational approach consists in simulating the dynamical system at a microscopic (or submacroscopic) level. Cellular automata are commonly used for this purpose. Space and time are discretised. A grid of interconnected cells is introduced where each cell describes a small volume in space. The evolving properties are asso

The computational description is not fully microscopic (this would be completely unfeasable from the viewpoint of simulation because billions of particles are involved). Instead, each particle is in fact a collection (e.g. 20,000) of actual particles. Nevertheless the phenomena that one expects in fluid flow, such as diffusion, wave propagation, emerging vortices, etc. can be convincingly synthesised. (Fig. 7).

There is a growing amount of activity in many areas of physics to use such a computational approach (see the new

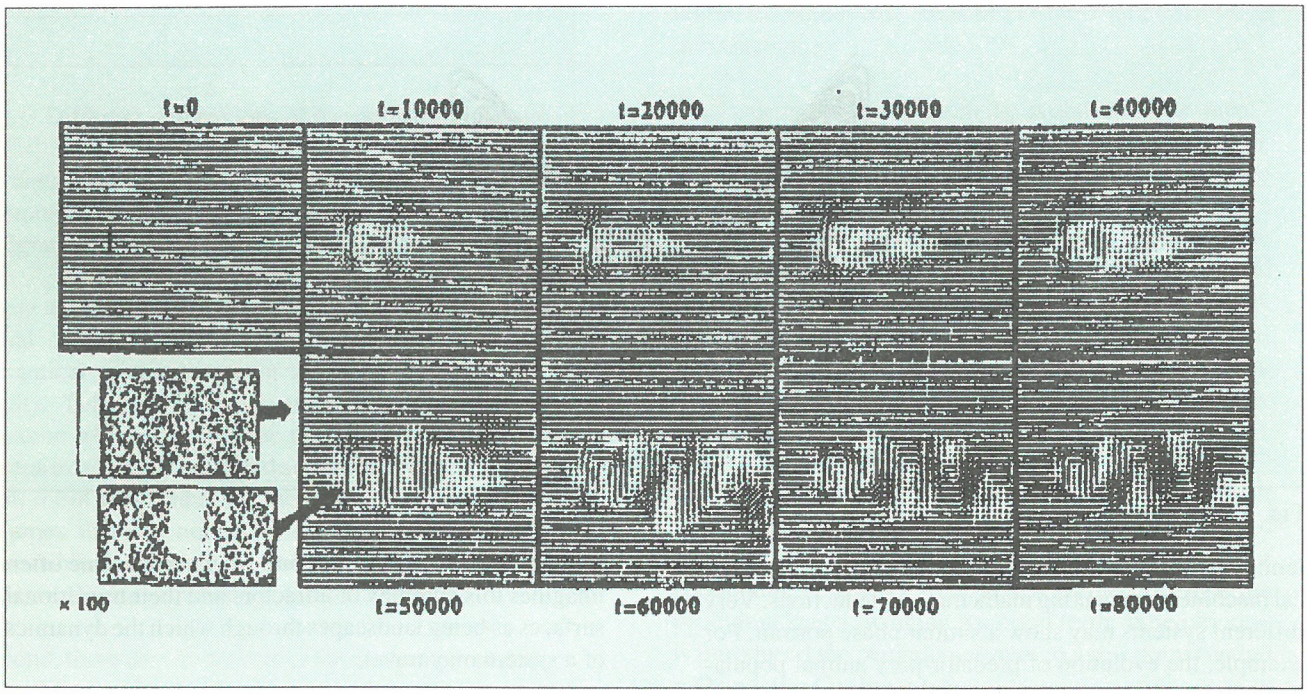


Fig. 7 Hydrodynamic flow simulated by lattice gas on Connection Machine.

ciated with each cell as state-variables. Derivatives in differential equations are replaced by computational rules associated with each cell that update the state-variables at discrete time steps.

A classical example of this approach is the modelling of phenomena in fluid dynamics by lattice gases [18]. The cells in lattice gases are hexagonal or triangular. They can be occupied by particles. Particles move around between cells. The computation rules associated with each cell describe what happens when particles meet each other or another obstacle. One example of such a rule is the following:

If two particles enter a cell from opposite sides, they will exit through a different pair of opposite walls.

journal Complex Systems for a representative sample of this work). There appear to be substantial advantages compared to partial differential equations [67]. All this work is also relevant for the use of complex dynamics in Artificial Intelligence. It clearly shows the possibility to synthesise the emerging properties of complex dynamical systems using computational models. If it can be done for physical phenomena, it can also be done for cognition.

2.3 RelaxationAnd Symmetry Breaking

Relaxation is the process by which a dynamical system evolves from a disturbed state to its natural state of stability given specific environmental conditions. The pendulum with friction will move to a vertical position after having been disturbed. A system of identical gas molecules will relax towards a state where there is a

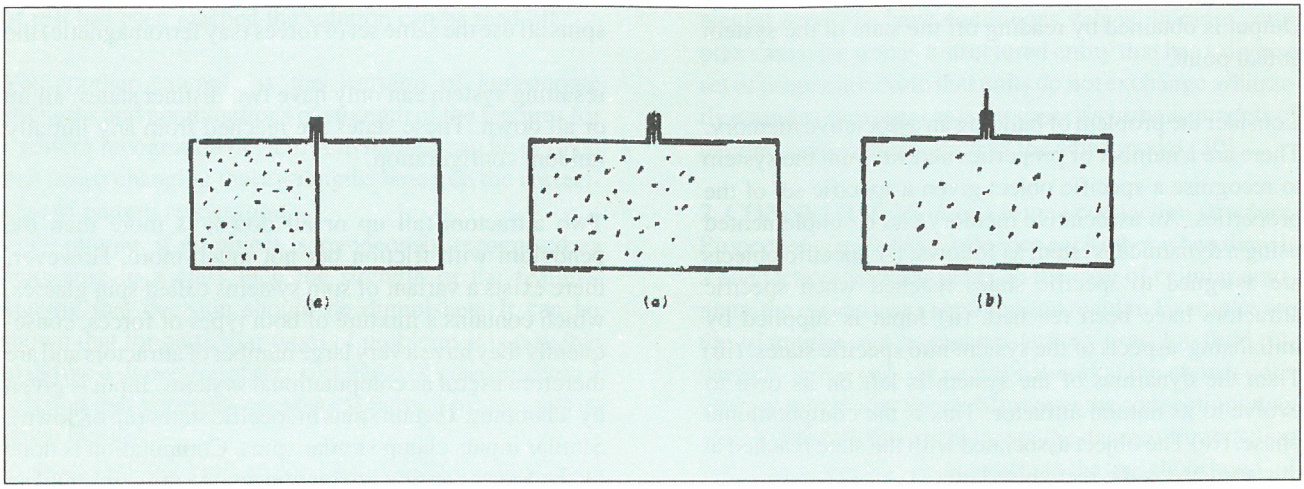


Fig. 8 Evolution towards random distribution.

uniform random distribution of the molecules, even if they were located in one partition of the space at the initial state (Fig. 8).

The relaxation depends on the laws governing the system. These laws are in many cases a function of the energy in the system and thus are related to the temperature of the environment.

An important distinction can be made between two types of behaviors. Systems may be brought in initial conditions and left undisturbed to reach an equilibrium state. For example, a boiling volume of liquid left undisturbed will cool off until it is in equilibrium with the environment. This behavior will be called equilibrium behavior. Second, systems may be continuously disturbed. They will build a new structure in response to this disturbance.

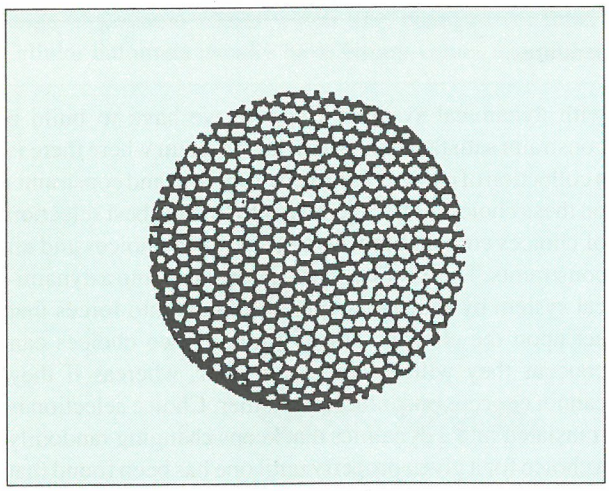


Fig. 9 Benard cells in uniformly heated dense liquid.

This structure will disappear as soon as the disturbance itself disappears. For example, when dense liquids are slowly and uniformly heated, a hexagonal grid composed of Benard cells will form (Fig. 9). This grid resembles a honeycomb structure.

This behavior will be called **non-equilibrium behavior**, because the dynamical system is kept continuously out of its equilibrium state. Equilibrium behavior leads to a state of symmetry, e.g. all molecules are evenly distributed in the available space. Non-equilibrium behavior leads to symmetry breaking. Local (so called dissipative) structures are formed in response to the continuous environmental disturbance [53].

3. DYNAMICAL COMPUTATION SYSTEMS

We now come to the next topic of the paper: how can complex dynamical systems be used as computational devices.

3.1 ComputingWith Attractors

Let us call dynamical systems used as computational devices dynamical computation systems. To build a computational device we must have the capability to represent information internally, to give inputs, to do computation, and to obtain output. Information can be represented in a dynamical system by associating it with specific internal states. Input is given by bringing the system in a specific environment. This includes not only global settings such as temperature, but also local settings. Computation is done by letting the system act according to its own natural dynamics, until the configuration has reached the attractor corresponding to the specific environment.

Output is obtained by reading off the state of the system at that point.

Consider the problem of building an associative memory. There are a number of properties and we want the system to recognise a specific object given a specific set of the properties. An associative memory can be implemented using a dynamical system as follows: (i) Specific objects are assigned to specific states reached when specific attractors have been reached. (ii) Input is supplied by initialising aspects of the system into specific states. (iii) Then the dynamics of the system is left on its own to evolve to its natural attractor. This is the computational phase. (iv) The object associated with the state reached at this attractor is the recognised object.

Note that we need at least as many attractors as objects to be recognised. A pendulum with friction has only one point attractor so it cannot represent anything. Whatever the initial input, which would take the form of introducing a force starting motion, that attractor would be reached. A more useful natural dynamical system from the viewpoint of information processing are spin systems [37].

Spin systems are models of magnets. They consist of a large collection of spins organised in a lattice. A spin can be up or down. At high temperatures spins keep changing direction randomly. When the majority of spins are in a specific direction, a magnet is formed in that direction. This happens at a critical low temperature. (Fig 10.)

The forces interacting among the spins determine the dynamics of the system. These forces come in two types: they either cause alignment of the spins (ferromagnetic) or do just the opposite (anti-ferromagnetic). When the spins all use the same set of forces (say ferromagnetic) the resulting system can only have two distinct states: all up or all down. These states are reached from any initially random configuration.

Two attractors (all up or all down) is more than the pendulum with friction but not much more. However, there exists a variant of spin systems called spin glasses, which contains a mixture of both types of forces, consequently they have a very large number of attractors and are therefore useful as computational systems. Input is given by 'clamping' certain spins in specific states (up or down). Similar inputs clamp similar spins. Computation is done by the forces causing interaction among the spins until an equilibrium state is reached. At that point spins are in specific positions and this configuration represents the object associated with the given input.

Associative memories are not the only thing one can build with dynamical systems. Suppose we have to build a constraint satisfaction system, i.e. a system where there is a collection of choices for given properties and constraints on these choices. The objective is to find the best selection of choices compatible with a set of given choices and all constraints. This problem can be translated into a dynamical system by translating the constraints into forces that act upon the choices. For example, if two choices can cooccur they will enforce each other, whereas if they cannot cooccur one inhibits the other. Choice selection is translated into a dynamics that keeps changing randomly a choice for a given property until one has been found that is compatible with the others. The initial choices are input by setting certain choices fixed at the beginning. Then the dynamics of the system is left to develop and when a state

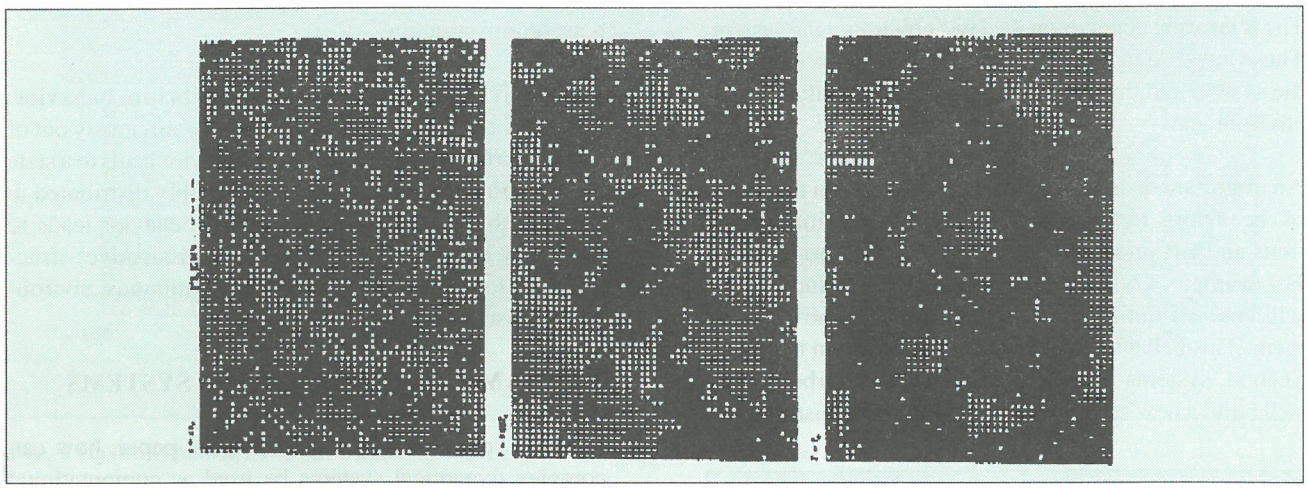


Fig. 10 Random distribution, and spin-alignment at different temperatures.

of rest has been reached the solution can be read off.

Yet another example is the learning of appropriate strengths and combinations of strength of the features for a pattern recognition device. A dynamics can be defined that keeps changing these strengths based on the correctness of pattern recognition.
For example, if an object is erroneously recognised as belonging to a class then the strength of the features making that decision should be diminished. It can be shown that for particular types of problems it is possible to define a dynamics of this sort that will converge into a solution that correctly connects features with the appropriate classes.
In most systems built so far point attractors have been used because they are the easiest to design or study. But it is in principle possible to use other types of attractors. Proposals exist even for using chaotic dynamics (see e.g. [51]). They will not be discussed further in this paper.

3.2 Definition Of Dynamical Computation Systems

Given that it is possible to do computation with dynamical systems, it becomes worthwile to formulate abstract models which can be the basis of implementations in computational media. A large variety of such models now exist. They range from all sorts of cellular automata [67], [72], to spin systems [37], and various forms of neural networks and connectionist systems [56]. Nevertheless it is possible to identify a general framework of which most of these models are variants. This framework has the following components:

1. UNITS AND PROPERTIES: A dynamical computation system consists of a finite collection of units each with a finite number of properties. The properties have a range of possible values which may change over time. Cellular automata usually have binary values (0 and 1). Neural networks have continuous values. In more complex cases, the unit is a structured entity that has a diverse set of properties. Note that units do not exchange arbitrarily complex messages as is the case of symbolic models of parallel computation, such as the actor model [30].

2. CONNECTIONS: Units form a connection structure. Properties of units may influence each other when there is a connection between them. In the case of cellular automata, the connections are fixed and regular. Examples are the triangular and hexagonal lattices used for fluid dynamics. In the case of neural networks, the connections form an irregular network. Moreover the connections may vary in strength thus influencing the degree of interaction between properties. In spin systems the 'neighborhood' of a unit, i.e. the units to which a unit is connected may vary but the units are fixed in a regular grid.

3. COMPUTATION RULES: The evolution of the value of the properties is based on a finite set of computation rules. The rule determines the new value of a property p based on the set of properties with which p is connected. Computation is therefore local. There is no way to get a global impact on all properties at once. The rules may be fixed or variable and evolving themselves. They are usually uniform over the complete system or dependent on a specific region or on the type of units or properties.

4. OPERATION: All rules are applied to all properties simultaneously. A dynamical computation system is therefore massively parallel. The execution of rules may be synchronised. All values for all properties are then computed before the next set of values is computed. Some models, particularly for connectionist systems, require however that the system behaves in an asynchronous way.

It has now been demonstrated quite convincingly that

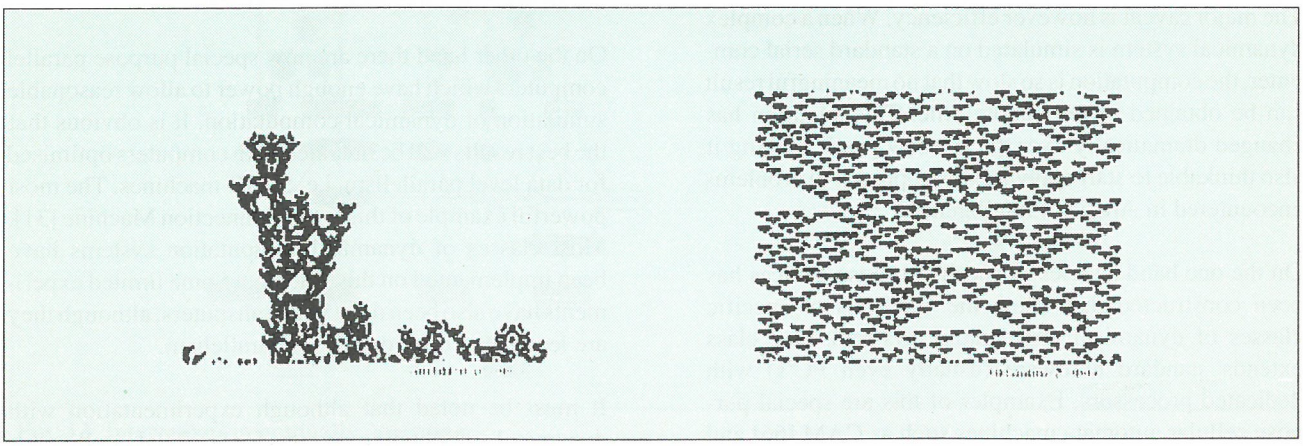


Fig.11 a. chaotic dynamics. b. complex local structure in cellular automata.

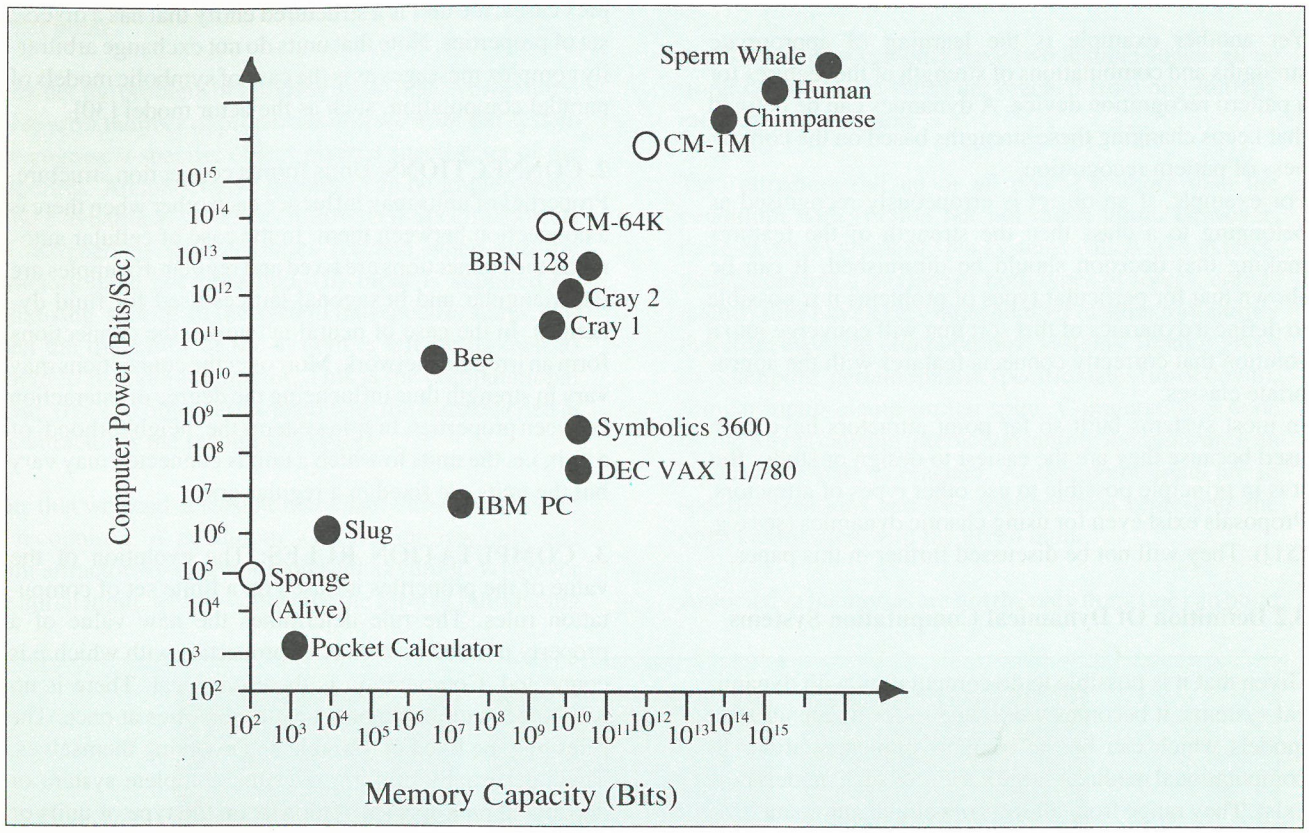


Fig.12 Comparison of brain power with hardware (from H. Moravec, CMU).

dynamical computation systems are capable of all the complex behavior we associate with natural dynamical systems, including all known types of attractors, phase transitions at critical values, self-organisation through the formation of dissipative structures, etc. For example, [72] has shown that even in one-dimensional synchronous cellular automata strange attractors or local complex structures may arise.
The major caveat is however efficiency. When a complex dynamical system is simulated on a standard serial computer, the computation is so slow that no meaningful result can be obtained in reasonable time. This situation has changed dramatically over the last few years, making it also thinkable to start applying this approach to problems encountered in Artificial Intelligence.

On the one hand a series of special purpose devices has been constructed that allow the synthesis of specific classes of dynamical computation systems. One class extends standard hardware (usually even PC's) with dedicated processors. Examples of this are special purpose cellular automata machines such as CAM [66] and RAP [9], and neural networks processors such as the ANZA Neurocomputing Coprocessor System, or the SIGMA-0 and Sigma-1 Neurocomputer workstations. A second class uses VLSI. This allows the construction of very fast implementations often exploiting the dynamics of the hardware to mimic the desired dynamics of the computational system. This approach has been used to implement the Hopfield net and to build an artificial retina based on neural network models [43].

On the other hand there are now special purpose parallel computers which have enough power to allow reasonable simulation of dynamical computation. It is obvious that the best results will be obtained with computers optimised for data level parallelism, i.e. SIMD machines. The most powerful example of this is the Connection Machine [31]. Most classes of dynamical computation systems have been implemented on this machine. Some limited experiments have also been done with transputers, although they are less adapted for data level parallelism.

It must be noted that although experimentation with dynamical computation systems for artificial intelligence has now become feasible, several orders of magnitude of

4. DYNAMICAL COMPUTATION SYSTEM FOR ARTIFICIAL INTELLIGENCE

The major class of dynamical computation systems explored so far for AI are neural networks. They are a recent revival of ideas that were a dominating research paradigm in the sixties until major obstacles were discovered [48]. Neural networks come in two basic types: spreading activation networks, and relaxation networks. Neural networks mostly exhibit equilibrium behavior. An alternative class are genetic algorithms, pioneered by Holland [34], and based on the same principles as natural selection. They constitute a clear example of non-equilibrium behavior. Another more recent class are reaction-diffusion models [64].
These different computational models are briefly discussed in the coming sections.

4.1 Spreading Activation Networks

Neural networks consist of units linked together in a network. Each unit has two properties: an activation value and a treshold. The activation value is a continuous value. The connections between the units have a weight which can be positive (in which case the connection is said to be excitatory) or negative (in which case the connection is inhibitory). In spreading activation networks, also called feed-forward networks, the connection between two cells is asymmetrical. Although there may be loops in the network, propagation basically flows in one direction. Some units are viewed as inputs which receive directly information from the environment. Other units at the end of the propagation wave are viewed as outputs. Spreading activation networks are therefore mostly used as associative memories or recognition devices, although they have also been used to encode various forms of inference [21].

4.1.1 Operation Of Spreading Activation

The perceptron [48] is a typical example of a spreading activation network. The propagation rule is a linear combination of the weights and inputs based on the following formula.

$$p(u,t1)=i(u_1\ t_0)xw(u_1\ t_0)+.....+i(u_0\ t_0)xwi(u_n\ t_n)$$

p(u,t) is the level of activation of a unit u at time t. i(u,t) represents the input from unit u at time t and w (u,t) represents the weight of the connection between u and p at time t. The output of a unit is determined by the following law.

$$o(u,t)=1 \text{ if } p(u,t)>\theta\ (u,t),\ 0 \text{ othetwhise}$$

$\theta(u,t)$ is the threshold of u at time t. Other types of spreading activation networks may employ different formula for weight combination or propagation but they follow the same schema.

When there are no loops in the network, it is easy to see that the system will always reach an equilibrium state in a straightforward manner. When there are loops more complex behavior, such as the formation of coalitions [24], or the gradual domination of one interpretation may emerge (see Fig. 13 from [70]).

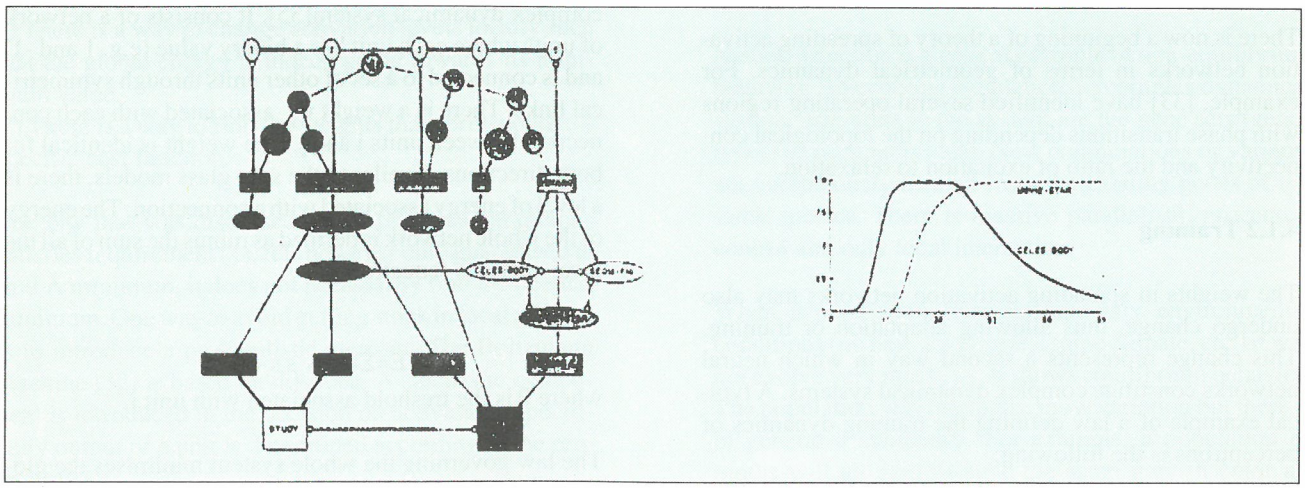


Fig. 13 Interpretation gradually emerging.

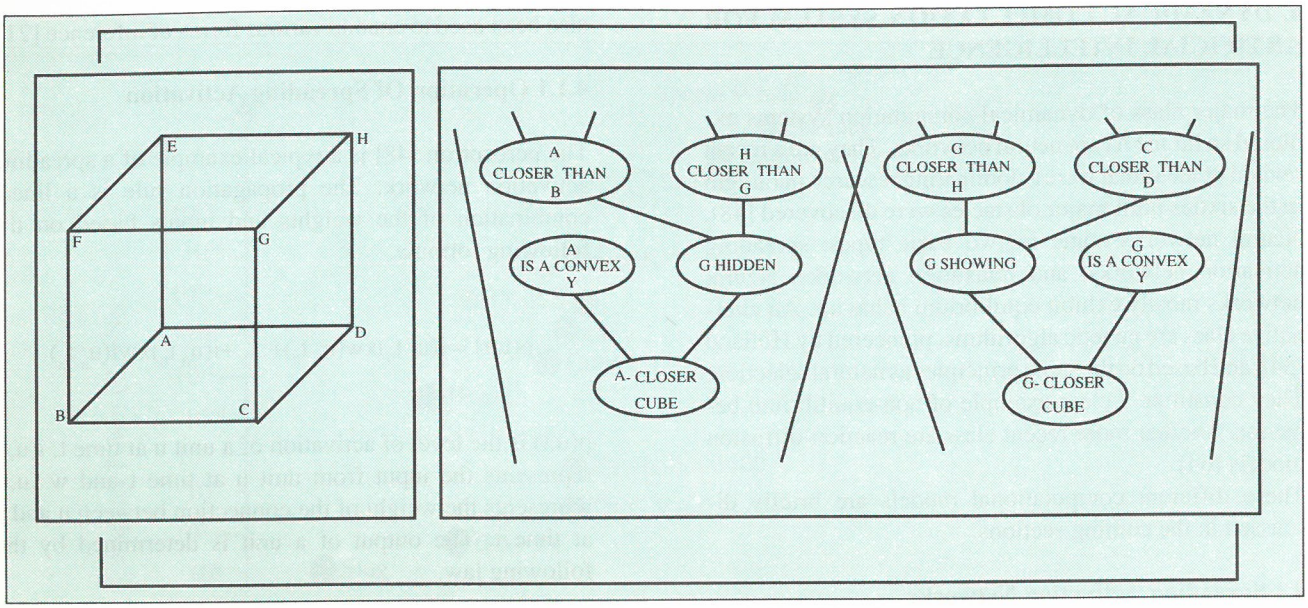


Fig. 14 a. Necker cube. b. Neural network for Necker cube.

One example of these phenomena is the Necker cube, an optical illusion in which a cube drawing may be interpreted two ways (Fig.14a). Either point G is closer or point A is closer. This phenomenon can be modeled using a neural network such as contained in Fig.14b Straight lines indicate excitation (i.e. positive weights) and lines with circles indicate inhibition (i.e. negative weights). The units which belong to one interpretation will enforce each other but at some point the system may flip to another interpretation in which case the previous interpretation is inhibited and the elements of the other interpretation enforce themselves. In the language of dynamical systems theory we would say that the dynamics keeps switching between the two interpretations thus exhibiting a limit cycle attractor.

There is now a beginning of a theory of spreading activation networks in terms of geometrical dynamics. For example, [33] have identified several operating regions with phase transitions depending on the topological connectivity and the ratio of excitation to relaxation.

4.1.2 Training

The weights in spreading activation networks may also undergo change, thus allowing adaptation or training. This change represents a second way in which neural networks constitute complex dynamical systems. A typical example of a law defining the training dynamics of perceptrons is the following:

$$\Delta w_{ij}=n(t_i-a_i)a_i$$

The perceptron convergence theorem [48] proofs that this law yields a stable and correct assignment of the weights after a finite series of training examples. In other words, the training dynamics leads to a point attractor. This powerful result is however limited to networks without intermediate units between input and output. Important breakthroughs have been obtained recently with the discovery of new training laws that allow the learning of weights for hidden units also. One example of this is back-propagation [56].

4.2 Relaxation Networks

The first major example of a relaxation network is the Hopfield net. The Hopfield net is an abstract model of a complex dynamical system[35]. It consists of a network of units where each unit has a binary value (e.g. 1 and -1) and is connected to a set of other units through symmetrical links. There is a weight $w_{i,j}$ associated with each connection between units i and j. The weight is identical for both directions. Similar to the spin glass models, there is a level of energy associated with a connection. The energy of the whole network is defined as minus the sum of all the local energies (equation 1).

$$E=\Sigma i<j\ w_{ij}s_is_j+\Sigma\theta_is_j$$

where θ_i is the treshold associated with unit i.

The law governing the whole system minimises the globalenergy by minimizing local energies. This law is as

follows, assume

$$E_{local} = \Sigma\ i<j\ w_i jbs_i\ s_j$$

If E_{local} is less than the treshold, then s_i is set equal to -1, otherwise it is set equal to 1.

It is easy to see that given this law the contribution to the total energy of an individual unit will diminish until a stable state of miminal energy is reached. The law depends on temperature which implies that if the temperature is changed again, the systems goes back in action until a new equilibrium state is reached. The minimal energy state acts as a point attractor. Once this state has been reached the system remains stable and will no longer change, even if local disturbances are introduced.

Relaxation networks resemble spreading activation networks. They also consist of units connected to each other through weighted links. The main difference is that all units are active at all times. Relaxation networks are therefore much closer to natural dynamical systems than spreading activation networks. Relaxation networks have been used mainly as constraint satisfaction systems.

Optimisation problems, and this includes also content-addressable associative memories or search problems like the traveling sales man problem, can be analysed in terms of a function that needs to be minimised. This function, usually called the cost function, expresses the characteristics of the best solution.

The basic idea behind Hopfield nets is to use this function as the energy function of the net and to have a dynamics where the activation of the units is such that the energy function is minimised. The idea can only work when
1) There is a way to change activation levels locally such that the global energy is indeed going towards its minimum value.
2) There is a way to find the weights that correspond to a specific cost function.

The law that was discussed in the previous paragraphs satisfies requirement (1). It is however only guaranteed to find A minimum, it does not necessarily find the deepest minimum. One way to avoid getting stuck in local minima is to introduce a probabilistic element. The Boltzmann machine [32] is based on this idea. A stochastic component is introduced in the decision on activation. Specifically output of a unit is determined according to the probability

$$\frac{1}{1+e^{-\Delta \frac{E(s)}{T}}}$$

This stochastic component is linked to a global parameter (usually called the temperature by analogy with physical systems). With high temperature, the system behaves only partially according to the rules given earlier on. This allows it to escape from local minima. By bringing down the temperature, there is less randomness and the rules are followed more strictly. The theoretical background for these principles are derived from algorithms for simulating annealing. [38]. Note that the Boltzmann machine still exhibits an equilibrium behavior with a point attractor for a given initial environment after which the state of the units determines the output of the computation.

Any kind of problem that can be formulated as a constraint satisfaction problem can be translated in this paradigm. This has been done for example for natural language parsing [57] and even unification [3]. In most cases this translation is done by hand. A cost function is derived from the optimisation problem itself. This cost function can then be transformed to find appropriate values or the weights.

However, just as for spreading activation networks, a training dynamics can be defined for relaxation networks which allows the system to find its own weights. One algorithm, discovered by Sejnowski and Hinton [32] is similar to the delta rules developed for perceptron like networks. The basic idea is to have two phases. In the first phase, input and output vectors are presented and the weight incremented between active units. In the second phase, only the input vector is given and the weight is decreased. It can be shown that this process will eventually lead to appropriate weights reflecting the given pair of input and output vectors.

4.3 Genetic Algorithms

Natural selection is a typical example of non-equilibrium behavior. An evolving population is a complex dynamical system. The units in the system are the gene sets (genotypes) of each individual. New combinations or variants are continuously formed from the existing genes of the same species. There is massive parallelism, no central control and only local interaction.

When a population lives in stationary environmental conditions (no new adversaries, same climate, etc.) it will not evolve very much, i.e. it shows equilibrium behavior. The population will internally show variation but there is in general a symmetry. For example, a population of moths will all tend to have the same greyish color although some may tend to white and others to dark. When one feature of the population undergoes pressure, normal fluctuations will start to be enforced leading to a symme-

try breaking. For example, if the color of the trees on which the moths live becomes darker, there will be an evolution towards a dominating darker population [40].

Natural selection can be viewed as a search process. Given that search is a fundamental component of many AI tasks, this mechanism can be used as a model for doing search, and therefore also constraint satisfaction. This is what genetic algorithms do.

There is a representation in terms of bit strings of the problem (by analogy with genes and chromosomes). There is a way to generate an initial population of solutions and an evaluation function that performs the role of the environment. It rates solutions according to how successful they are towards a solution of the problem. There are genetic operators that change solutions. One example of such an operator is random mutation. Another one is inversion, altering the location of genes on a chromosome. The most important operator however is crossover which allows bringing together two partially successful solutions into a better solution.

Since its conception, genetic algorithms have been applied quite succesfully to a wide variety of problem domains. Examples include the optimisation of a communication network [13], scheduling [14], selection of mapping of typewriter keys [28], etc. The selectionist approach underlying genetic algorithms has been shown to be superior to the connectionist approach to constraint satisfaction problems as exemplified by the Hopfield net [8].

4.4 Reaction-DiffusionModels

Based on the observation that many biological dynamical systems, particularly those exhibiting self-organisation, involve reaction and diffusion, Steels [62] recently formulated a computational model based on reaction-diffusion systems. Turing himself in fact already developed the mathematical foundation for such systems and showed in particular that they are capable of non-equilibrium behavior.

Turing [69] proposed a simple model showing convincingly how non-equilibrium behavior may arise and lead to structure. He assumed a reaction-diffusion model involving two cells and two substances X and Y. X_1, Y_1 and X_2, Y_2 denote the concentrations of X and Y in cell 1 and 2 respectively. The permeability coefficients accross the membranes connecting the two cells are P_x=0.5 and P_y=4.5. The change in X and Y in cell 1 is given by the following equations

$$\frac{dX_1}{dt} = 5X_1 - 6Y_1 + 1 - 0.5(X_1 - X_2) \qquad (2)$$

$$\frac{dY_1}{dt} = 6X_1 - 7Y_1 + 1 - 4.5(Y_1 - Y_2) \qquad (3)$$

Similarly for the other cell.

Next, Turing made the following observation: when $X_1=X_2=1$ and $Y_1=Y_2=1$, $dX_1/dt=0$ and $dY_1/dt=0$. So there is a stationary homogeneous solution. X and Y are evenly distributed over the two cells. In this case the system shows an equilibrium behavior leading to a symmetrical structure.

Next consider that small shifts, possibly due to thermal fluctuation, occur. As we would expect in equilibrium behavior, these shifts are damped. For example, if X_1=0.5 and Y_1=0.8 we get (4) and (5)

$$\frac{dX_1}{dt} < 0 \quad \text{for } X_2 < 7.7 \qquad (4)$$

$$\frac{dY_1}{dt} < 0 \quad \text{for } Y_2 < 1.1 \qquad (5)$$

These negative rates will bring the system back to its stationary equilibrium state.

But consider now what happens when the fluctuation gets larger. For example, if X_1=1.06, Y_1=1.02, X_1=0,94 and Y_1=0.98. Now dX_1/dt=0.12 and dY_1/dt=0.04. These rates are positive indicating that matter accumulates in cell 1 and dissipates in cell 2. Thus the symmetry of the homogeneous distribution is broken and we have an inhomogeneous distribution which enforces itself. The example illustrates that under certain conditions a system may switch to non-equilibrium behavior, at which point it reorganises itself to form a new structure.

An amazing range of phenomena is based on this fundamental principle of symmetry breaking. First of all phenomena of self-organisation and the formation of dissipative structures in physics and chemistry can be modeled this way. See e.g. [53], [29]. Also many fundamental mechanisms in biology such as natural selection, pattern formation, even the origin of life have been modeled in terms of reaction diffusion systems showing symmetry breaking. See e.g. [20], [44], [2].

A computational reaction-diffusion system consists of a cellular grid. Each cell may contain a number of symbols with varying strengths. A cell also contains reaction rules which regulate how the strength of various symbols evolves in time. For example, one symbol may activate or inhibit the strength of another symbol. Each symbol has also a diffusion behavior defined by diffusion coefficients.

The exploration of the reaction-diffusion model for Artificial Intelligence is still in its early phases. They appear to be well suited for dealing with problems involving the construction, maintenance and adaptation of structures (such as networks). At the same time, selectionist phenomena and therefore search and constraint satisfaction can easily be implemented. Steels [64] gives examples of how reaction-diffusion can be used in common sense and problem solving.

5. NEW PERSPECTIVES ON KEY ISSUES

At this point we are only at the beginning of the exploration of the dynamics paradigm. Nevertheless it is already clear that it will have a major impact on Artificial Intelligence. One way to view the relation between current AI and the dynamics approach is as a relation between macroscopic and microscopic sciences. Notions like symbol, goal, rule, frame, control structure, deduction chain, etc. may turn out to be macroscopic ways of describing cognitive systems. As we know from chemistry and physics macroscopic descriptions are important and may lead to powerful theories.
The dynamics paradigm appears to develop the microscopic counterpart of AI and theories on how macroscopic cognitive phenomena emerge from microscopic laws.
Compare this to thermodynamics (the macroscopic science), particle physics (the microscopic science), and statistical mechanics (the bridge between the two). The interest in developing a microscopic science and technology for AI is that certain phenomena may not be synthesisable at the macroscopic level.

In this final section various key issues of Artificial Intelligence are briefly examined from the viewpoint of the dynamics paradigm. The issues are representation, reasoning, search, and learning. Many of the observations are conjectures and speculations concerning research in progress.

5.1 Representation

It is obvious that the ability to perform reasoning presupposes representations of the domain. This has lead to a large emphasis in Artificial Intelligence research on questions concerning knowledge representation. One of the early key hypotheses was that the symbol is the primary unit of representation. Based on this symbolic level, various formalisms have been implemented, ranging from logic-based formalisms like Horn-clause logic [39], to semantic networks [54] or frame-based representations [7]. The success of current AI techology rests largely on the capabilities of these representational formalisms.

The dynamics paradigm assumes very different sorts of representations. First of all they are subsymbolic and distributed. A concept is no longer associated with a symbol, i.e. a fixed entity that is manipulated as such.

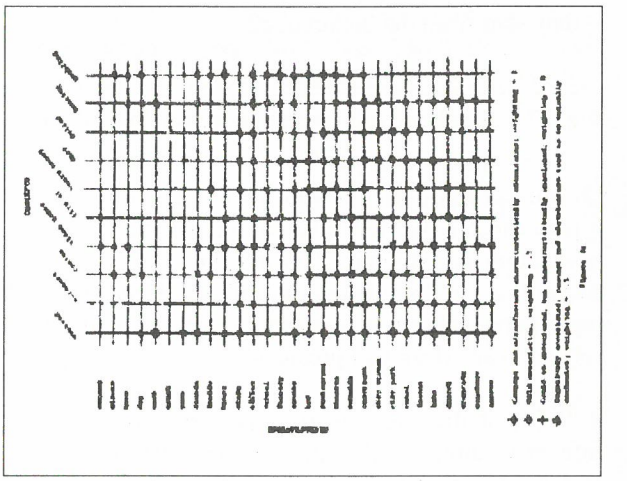


Fig. 16 Semantic marker set active when specific concepts are active

Instead a concept is represented as diffuse set of more elementary objects distributed in space. These more elementary objects constitute in a sense the definition of the concept. For example, in the language parsing experiments of Waltz and Pollack [70] each concept was represented as the diffuse activation of primitive semantic markers (Fig 16).

Subsymbolic representations offer a number of remarkable properties and advantages over their symbolic counterparts. Here are some examples:

1. A symbol may be present to a certain degree. Subsymbolic representations therefore constitute an alternative to the numerical approaches for dealing with uncertainty. This is particularly useful in interpretation tasks. For example, the competition between two hypotheses can be dealt with by having two competing symbols until one symbol starts to dominate (observe again the formation of coalitions in the parsing experiment,Fig. 13).

2. Because activation of a symbol is equal to activation of its defining parts, subsymbolic representations offer an immediate solution to the problem of building associative memories. This problem has always been avoided in many AI systems. If it was tackled large amounts of knowledge were presupposed [58].

c1 and c2 are circles
c1 is smaller than c2
c1 is on top off c2
11, 12, 13 and 14 are lines
11 and 12 start from the
left and right sides off c2
13 and 14 are parallel
they start from the botton of c2

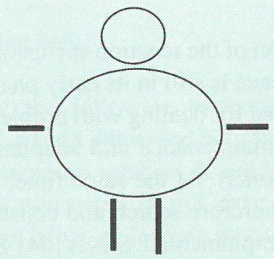


Fig. 17 a. Symbolic representation. b. Analogical representation.

3. Another key difficulty with symbolic representations is partial matching, including finding analogies. If the representation is symbolic, this matching must be done sequentially and spelled out in full detail. Subsymbolic representations can be compared in parallel and can be related to any degree of precision.

4. Subsymbolic representations perform basic forms of inference automatically. An example is the follow up of ISA links important for implementing inheritance. Because concepts share large subsets of component elements, activation of a concept at a low point in a ISA hierarchy automatically activates more general concepts.

The dynamics paradigm has also focused the attention on analogical representations [11]. An analogical representation preserves key features of the object to be represented implicitly in the representation. It is different from a symbolic representation where the key features are decoded in symbols that have no correspondent to the represented object. Compare the two representations of Fig 17.
Although there has been some work on analogical representations connected to propositional knowledge in the classical AI literature (e.g. [27]), analogical representations are seldom used today, mostly because of the difficulty to implement them in a symbolic environment them and to interface between analogical and propositional knowledge [26]. The exploitation of analogical representation is a natural thing to do for dynamical computation systems. One example is the artificial retina built by Mead, et.al. [43] in which spatial features are maintained in the underlying representation.

5.2 Search

Since the empirical studies of Newell and Simon [49], it is recognised that most mental problems involve some form of search. Moreover this search is in most cases combinatorially explosive [52], or is to be avoided for other reasons, e.g. the increasing cost of observations. Search has therefore become one of the key problems in Artificial Intelligence. Initially search used to be performed with weak methods, such as breadth-first search or hill-climbing. More recently there is a consensus that domain-dependent heuristics, either implemented in the form of heuristic rules or through meta-level control of inference, are more powerful and necessary. This insight is one of the key principles underlying expert systems [61].
Although domain-dependent control of search is obviously much more effective, it still has problems. For one thing, it has turned out to be very difficult to extract the explicit control strategies and heuristics from the expert suggesting that not all of them are explicitly represented in their own mind. At the same time, explicitly programmed control structures are still very brittle. They only work within a definite scope of cases. When problems occur that fall outside the boundaries, performance drops dramatically [60].

Complex dynamical systems offers a different paradigm for approaching search. The basic idea is to do the search, i.e. use brute force. The nature of the search is however very different from symbolic search. The search space is defined over subsymbolic representations of the domain and randomness is used to do the exploration. Control over the search process is not preprogrammed or explicitly represented (as is now very common with meta-reasoning capabilities). Instead it emerges by the underlying dynamics of the search process.
Two examples of this approach were already discussed briefly earlier on: constraint satisfaction through relaxa-

tion in the Boltzman machine and selectionist search in Holland's genetic algorithms. What is remarkable in both cases is that a solution is found, even where a symbolic search is very unlikely to find a solution, and even though there is no explicit control structure.

5.3 Reasoning

The symbolic paradigm has made the implementation of complex logical deduction systems and intricate knowledge representation schemes possible. Although it has been shown how unification can be implemented in a dynamical system [3], the logical approach to reasoning does not seem the most natural formalism for dynamical computation systems. Instead, associative memories, and flexible matching over large amounts of cases are natural capabilities of dynamical systems which are difficult for the symbolic paradigm. In this sense models will be closer to production system architectures [49], or frame-based representation schemes [46] and society of mind theories [47].

5.4 Learning

The dynamics paradigm offers also a major new alternative to learning, namely self-organisation. Learning in the symbolic paradigm involves the definition of complex algorithms that carefully manipulate symbolic representations. Self-organisation is just the opposite. New structures emerge spontaneously without any central supervising agent that manipulates the representations. Examples are the training dynamics defined for spreading activation and relaxation networks. (See the overview in [25]).

Most work on learning in the dynamics paradigm has two characteristics: (i) it is inductive in nature, i.e. it detects similarities over large amounts of data similar to statistical induction, and (ii) it works on low level representations, e.g. bitmaps or directly observable features. In contrast learning within the symbolic paradigm operates mostly over high level descriptions. Moreover current methods, such as explanation-based learning, deductively exploit a theory of the domain.

6. CONCLUSIONS

The paper discussed a newly emerging paradigm in computer science in general and Artificial Intelligence in particular. The paradigm is based on the idea of viewing computation as a complex dynamical system. The paper introduced some elementary notions from dynamical systems theory and gave a brief overview of several classes of dynamical computation systems currently being investigated. The dynamics paradigm appears to hold enormous potential for explaining phenomena in intelligence and seems the only viable way to use massively parallel computers with millions of processors. On the other hand, a lot of research remains to be covered and dramatic technological advances, related to optical and molecular computing may be needed.

7. ACKNOWLEDGEMENT

I am indebted to Jo Decuyper and Bernard Manderick for helping me find my way in the new paradigm discussed in the paper.

8. REFERENCES

[1] Abrahams and Shaw, GEOMETRICAL DYNAMICS. (3 volumes), Aerial Press, Inc. Santa Cruz Ca., 1985

[2] Babloyantz, A., MOLECULES, DYNAMICS AND LIFE. AN INTRODUCTION TO SELF-ORGANIZATION OF MATTER,Wiley-Interscience. New York, 1986

[3] Ballard, D., PARALLEL LOGICAL INFERENCE AND ENERGY MINIMIZATION, Proceedings Of The National Conference On Artificial Intelligence, Philadelphia, August 1986, 203-208, (1986)

[4] Bell, T. E., OPTICAL COMPUTING: A FIELD IN FLUX, IEEE Spectrum., August 1986

[5] Berge, Pomeau and Vidal, L'ORDRE DANS LE CHAOS, Hermann, Paris, 1984

[6] Bienenstock, E., F. Fogelman-Soulie and G. Weisbuch., PROCEEDINGS OF THE NATO ADVANCED RESEARCH WORKSHOP ON DISORDERED SYSTEMS AND BIOLOGICAL ORGANIZATION. Springer Verlag. Berlin,1986

[7] Bobrow, D. and T. Winograd, AN OVERVIEW OF KRL, A KNOWLEDGE REPRESENTATION LANGUAGE, in Cognitive Science, Vol 1, n. 1. Lawrence Erlbaum Ass., New Haven,1977

[8] Brady, A SELECTIONIST APPROACH TO THE TRAVELING SALESMAN PROBLEM, Nature, 1986

[9] Clouqueur, A. and D. d'Humieres, RAP1, A CELLULAR AUTOMATON MACHINE FOR FLUID DYNAMICS, in Journal of Complex Systems, Vol 1. nr. 4, 1987

[10] Conrad, M., ON DESIGN PRINCIPLES FOR A MOLECULAR COMPUTER, in Comm. ACM. Vol 28 nr. 5,1985

[11] Churchland, P., SOME REDUCTIVE STRATEGIES IN NEUROBIOLOGY, Journal of Current Philosophy, 1986

[12] Davis, L., GENETIC ALGORITHMS AND SIMULATED ANNEALING, Research Notes in Artificial Intelligence, Pitman, London, 1987

[13] Davis, L. and S. Coombs, OPTIMIZING NETWORK LINK SIZES WITH GENETIC ALGORITHMS, in Elzas, et.al. (eds) Modelling and Simulation Methodology: Knowledge Systems Paradigms, North-Holland Pub. Co., 1987

[14] Davis, L. and F. Ritter, SCHEDULE OPTIMIZATION WITH PROBABILISTIC SEARCH, Proceedings of the 3rd IEEE Conference on Artificial Intelligence Applications,1987

[15] Dawkins, R., THE BLIND WATCHMAKER, Longman Scientific and Technical Pub. Harlow,1986

[16] Deneubourg, J., J. Pasteels and J. Verhaeghe, PROBABILISTIC BEHAVIOUR IN ANTS: A STRATEGY OF ERRORS?,in J. Theor. Biol, 1983

[17] Devaney, R., AN INTRODUCTION TO CHAOTIC DYNAMICAL SYSTEMS, Benjamin/Cummings Pub. Menlo Park, Ca.,1986

[18] D'Humieres, D. and P. Lallemand, NUMERICAL SIMULATIONS OF HYDRODYNAMICS WITH LATTICE GAS AUTOMATA IN TWO DIMENSIONS, Journal of Complex Systems, Vol 1 nr 4, Aug 1987

[19] Edelman, G., CORTICAL ORGANISATION AND THE GROUP-SELECTIVE THEORY OF HIGHER BRAIN FUNCTIONS, In Edelman and Mountcastle (eds.) The Mindful Brain, pp. 55-96, MIT Press, Cambridge, Ma.,1978

[20] Eigen, M., HOW DOES INFORMATION ORIGINATE? PRINCIPLES OF BIOLOGICAL SELF-ORGANIZATION, Advances in Chemical Physic. Vol XXXVII,For Ilya Prigogine, MIT Press. Cambridge Ma., 1978

[21] Fahlman, S., NETL: A SYSTEM FOR REPRESENTING AND USING REAL-WORLD KNOWLEDGE, MIT Press. Cambridge Ma., 1979

[22] Farmer, D. T. Toffoli, and S. Wolfram, CELLULAR AUTOMATA, Proceedings of an Interdisciplinary Workshop. Los Alamos, New Mexico,1983, North-Holland Pub. Co. Amsterdam, 1984

[23] Feigenbaum, E., THE ART OF ARTIFICIAL INTELLIGENCE: THEMES AND CASE STUDIES IN KNOWLEDGE ENGINEERING, 5th IJCAI, Cambridge. MA.,1977

[24] Feldman, J., NEURAL REPRESENTATION OF CONCEPTUAL KNOWLEDGE, Dept of Comp. Scie. University of Rochester,TR189, 1986

[25] Fogelman-Soulie, et.al., NETWORK LEARNING, in Kodratoff, et.al. Machine Learning, Springer-Verlag, Berlin,1987

[26] Gardin, F, Melzer B. and Stofell P., THE ANALOGICAL REPRESENTATION OF LIQUIDS IN NAIVE PHYSICS, Proc. ECAI 86, Brighton, 1986

[27] Gerlenter, H., J. Hansen and D. Loveland, EMPIRICAL EXPLORATIONS OF THE GEOMETRY THEOREM MACHINE, in E. Feigenbaum and J. Feldman. (eds) Computers and Thought,McGraw-Hill, New York,1963

[28] Glover, D., EXPERIMENTATION WITH AN ADAPTIVE SEARCH STRATEGY FOR SOLVING A KEYBOARD DESIGN/CONFIGURATION PROBLEM, Computer Science Department,Technical Report 86-03, University of Iowa, Iowa City, Iowa, 1986

[29] Haken, H., SYNERGETICS, AN INTRODUCTION. SPRINGER SERIES IN SYNERGETICS. Vol 1, Springer Verlag Berlin,1983

[30] Hewitt, C., VIEWING CONTROL STRUCTURES AS PATTERNS OF PASSING MESSAGES, Artificial Intelligence 8 (3), North-Holland Pub. Co., 1977

[31] Hillis, D., THE CONNECTION MACHINE,MIT Press, Cambridge, Ma.,1986

[32] Hinton, G. and T. Sejnowsky. LEARNING IN BOLTZMANN MACHINES, Cognitiva 85, Paris, 1985

[33] Hogg, T. and B.A. Huberman, ARTIFICIAL INTELLIGENCE AND LARGE SCALE COMPUTATION. A PHYSICS PERSPECTIVE, In Physics Reports,1987

[34] Holland, J., ADAPTATION IN NATURAL AND

ARTIFICIAL SYSTEMS, University of Michigan Press, Ann Arbor, 1975

[35] Hopfield, J., NEURAL NETWORKS AND PHYSICAL SYSTEMS WITH EMERGENT COLLECTIVE COMPUTATIONAL ABILITIES, Proceedings of the National Academy of Sciences USA, 79, 1982

[36] Jacquard, A., THE GENETIC STRUCTURE OF POPULATIONS, Springer-Verlag, Berlin,1974

[37] Kinzel, W., LEARNING AND PATTERN RECOGNITION IN SPIN GLASS MODELS, in Condensed Matter, Zetischrift fur Physik B. 60, Springer-Verlag, Berlin,1985

[38] Kirkpatrik, S., C. Gelatt, M. Vecchi, OPTIMIZATION BY SIMULATED ANNEALING, Science, 220, 1983

[39] Kowalski, R., LOGIC FOR PROBLEM SOLVING, North-Holland, Amsterdam, 1979

[40] Lewonton, R., ADAPTATION, in Scientific American, September 1978

[41] Maes, P., COMPUTATIONAL REFLECTION, VUB AI lab. TR-87-2, Brussels, 1987

[42] McCarthy, J., RECURSIVE FUNCTIONS OF SYMBOLIC EXPRESSIONS AND THEIR COMPUTATION MACHINE, Comm. ACM, 1960

[43] Mead, THE ARTIFICIAL RETINA, Proceedings of the 1st IEEE conference on neural networks, San Diego Ca., 1987

[44] Meinhardt, H., MODELS OF BIOLOGICAL PATTERN FORMATION, Academic Press, London, 1982

[45] Michalski, T., J. Carbonell and T. Mitchell (eds.), MACHINE LEARNING - AN ARTIFICIAL INTELLIGENCE APPROACH, Tioga Pub. CO., Palo Alto Ca., 1983

[46] Minsky, M., A FRAMEWORK FOR REPRESENTING KNOWLEDGE. In The Psychology of Computer Vision. P. Winston (ed.) McGraw-Hill Book, New York, (1975)

[47] Minsky, M., THE SOCIETY OF MIND, Basic Books, New York,1986

[48] Minsky, M. and S. Papert, PERCEPTRONS, MIT Press. Cambridge Ma.,1969

[49] Newell. A. and H. Simon, HUMAN PROBLEM SOLVING, Prentice-Hall, Englewood-Cliffs, New Jersey,1972

[50] Nicolis, J. and I. Prigogine, SELF-ORGANISATION IN NON-EQUILIBRIUM SYSTEMS, New York, Wiley, 1977

[51] Nicolis, J., CHAOTIC DINAMICS APPLIED TO INFORMATION PROCESSING, in Reports on Progress in Physics, Vol 49, n. 10, The Institute of Physics, 1986

[52] Pearl, J., HEURISTICS. INTELLIGENT SEARCH STRATEGIES FOR COMPUTER PROBLEM SOLVING, Addison-Wesley Pub. CO. Reading, Ma., 1984

[53] Prigogine, I., FROM BEING TO BECOMING. TIME AND COMPLEXITY IN THE PHYSICAL SCIENCES, Freeman and Company, New York, 1980

[54] Quillian, M. R., SEMANTIC MEMORY. in Minsky, M. (ed.) Semantic Information Processing, MIT Press,Cambridge, Ma., 1968

[55] Robertson, G., PARALLEL IMPLEMENTATION OF GENETIC ALGORITHMS IN A CLASSIFIER SYSTEM., in [13]

[56] Rumelhart, D., J. L. McClelland, et.al., PARALLEL DISTRIBUTED PROCESSING. EXPLORATIONS IN THE MICROSTRUCTURE OF COGNITION, 2 volls. MIT Press, Cambridge Ma., 1986

[57] Selman, B. and G. Hirst, PARSING AS AN ENERGY MINIMIZATION PROBLEM, in [13]

[58] Schank, R., DYNAMIC MEMORY, Cambridge University Press, Cambridge, 1985

[59] Shapiro, E., CONCURRENT PROLOG, MIT Press, Cambridge Ma., 1986

[60] Steels, L., SECOND GENERATION EXPERT SYSTEMS. FUTURE GENERATIONS COMPUTER SYSTEMS, North-Holland Pub. Co., 1984

[61] STEELS, L., THE DEEPENING OF EXPERT SYSTEMS, AI Comunications, Vol 0, 0. North-Holland Pub. Co. Amsterdam, 1987

[62] Steels, L., A SELECTIONIST APPROACH TO SELF-ORGANISATION, in Proceedings of the first IEEE Conf on neural networks, San Diego, Cal., 1987

[63] Steels, L., STEPS TOWARDS COMMON SENSE, VUB AI Lab. Memo 88-3, Brussels, 1988

[64] Steels, L., A REACTION-DIFFUSION MODEL OF COMPUTATION, VUB AI Lab. Memo 88-5. Brussels, 1988

[65] Sussman, G., HACKER, MIT AI Lab Tech. Rep, Cambridge, Ma.,1974

[66] Toffoli, T., CELLULAR AUTOMATA AS AN ALTERNATIVE TO (RATHER THAN AN APPROXIMATION OF) DIFFERENTIAL EQUATIONS IN MODELING PHYSICS, Physica 10D, North-Holland, Amsterdam, 1984

[67] Toffoli, T. and N. Margolus, CELLULAR AUTOMATA MACHINES. A NEW ENVIRONMENT FOR MODELING, MIT Press, Cambridge Ma., 1987

[68] Toulouse, G., S. DeHaene, and J.P. Changeux, SPIN GLASS MODEL OF LEARNING BY SELECTION, Proc. Natl Acad. Sci. USA, March 1986

[69] Turing, A., THE CHEMICAL BASIS FOR MORPHOGENESIS, Philos. Trans. R. Soc. London (B), 237, 1952

[70] Waltz, D. and J. Pollack, MASSIVELY PARALLEL PARSING, Cognitive Science, 1985

[71] Winston, P., LEARNING STRUCTURAL DESCRIPTIONS FROM EXAMPLES, MIT Techn. Rep. AI-lab. Cambridge, Ma.,1970

[72] Wolfram, S., STATISTICAL MECHANICS OF CELLULAR AUTOMATA, in Rev. Mod. Phys., Vol 55, n. 3 July 1983

[73] Woods, W., TRANSITION NETWORK GRAMMARS FOR NATURAL LANGUAGE ANALYSIS, Communications of the ACM, vol 13, n. 10, 1970

MODELLI QUALITATIVI PER IL SUPPORTO DECISIONALE

Mauro Cadei, Francesco Gardin
Dipartimento di Scienze dell'Informazione
Università di Milano

RIASSUNTO

In questo articolo viene descritto l'utilizzo di tecniche di ragionamento qualitativo quale parte integrante di un sistema di supporto alle decisioni.
L'idea di fondo dell'articolo é basata sull'osservazione che durante la fase di definizione di un modello nei Decision Support Systems (DSS) attuali possono non essere note alcune relazioni quantitative, come pure alcuni dati, mentre invece esiste una valutazione qualitative degli stessi.
La possibilità di estendere il formalismo espressivo di un DSS mediante l'introduzione di primitive qualitative, e del relativo meccanismo di calcolo associato, rende possibile la definizione di modelli decisionali puramente qualitativi o misti.
Il sistema realizzato in Prolog é basato su di una estensione della formalismo QSIM [10].
L'applicazione dimostrativa presentata è relativa alla scelta di prezzi di un prodotto a partire dall'utilizzo di un modello di mercato quantitativo-qualitativo.
Il ruolo di DSS del tipo di quello descritto nell'articolo é particolarmente indicato per agenti decisionali che operano ad un alto livello di astrazione del modello aziendale.

ABSTRACT

In this article we describe the use of qualitative reasoning techniques as a relevant part of a Decision Support System.
The basic idea on which this article is based is the acknowledgement that during the definition phase of current Decision Support System (DSS) models, some quantitative relation, as well as data, may not be known, whereas the qualitative version off the same are known.
The possibility of extending the DSS formalism by means of the introduction of qualitative primitives, and the associate computational engine, makes it possible to define a decision model which is either entirely qualitative or mixed.
The system has been implemented in Prolog and it is based on an extension of the QSIM formalism [10]. The application presented for demonstration purposes is about the definition of the appropriate price of a product using a mixed market model, i.e. qualitative and quantitative.The role of DSS of the kind described in this article is particulary suitable for decision agents who operate using a high level abstraction model of the company.

1. INTRODUZIONE

Numerose sono le interessanti prospettive aperte dalla ricerca nell'ambito dell'Intelligenza Artificiale (A.I.) per lo sviluppo di sistemi di supporto alla decisione (D.S.S.) [8], [2], [5], [6]. Quella da noi proposta è un'analisi del ruolo che può essere ricoperto dall'integrazione di nuovi strumenti espressivi, i modelli qualitativi [10] [11], in un generatore di D.S.S., dei vantaggi che ciò può comportare rispetto ai modelli tradizionali e però anche dei limiti che ne consigliano un uso diversificato, non alternativo, rispetto ad essi.

Tra le motivazioni che più hanno influito sulla nostra ricerca possiamo individuare l'interesse verso paradigmi che consentano un supporto nella generazione di un nuovo modello decisionale. Sentita è infatti, particolarmente nelle prime fasi del processo di modellazione, l'esigenza di strumenti espressivi a livelli concettuali meno dettagliati rispetto alle relazioni matematiche comunemente impiegate.

Interessante è pure la possibilità di offrire tali potenzialità espressive in fase di interrogazione del modello. Soprattutto ai livelli gerarchici più elevati del sistema decisio-

nale aziendale si rivela infatti comune l'esigenza di modelli caratterizzati da un'aggregazione molto elevata, che consentano un'adeguata valutazione della situazione attuale e delle strategie più opportune in termini molto generali, con caratteristiche di tipo qualitativo.

Un terzo ed ultimo motivo di interesse è costituito dalla constatazione di come un efficace impiego diretto di modelli da parte del decisore sia subordinato all'esistenza di un'interfaccia amichevole ma soprattutto alle capacità espressive del modello stesso che ne consentano una facile comprensione delle caratteristiche strutturali così come delle condizioni di applicabilità. E' chiaro quindi l'interesse verso forme di modellazione che realizzino, secondo la filosofia dei D.S.S., una rappresentazione espressiva che favorisca un rapporto diretto tra decisore ed implementazione del modello, senza il tramite di un intermediario, a tutto vantaggio di una maggiore versatilità ed efficacia.

Dopo un'introduzione ai modelli qualitativi, con particolare riferimento alla loro interpretazione espressa da Kuipers, la simulazione qualitativa, è qui illustrata la nostra proposta relativa ad un suo impiego in ambito decisionale. Alla luce delle importanti questioni, ampiamente sottolineate in letteratura, relative alla realizzazione di un supporto automatico al processo decisionale in azienda, individuiamo dapprima delle opportune modifiche ad essa, quindi le specifiche, soprattutto in termini di interfaccia con l'utente, per un sistema che la realizzi. Esse trovano espressione nel prototipo da noi sviluppato, una sorta di generatore di fogli elettronici qualitativi, le cui funzionalità sono saggiate attraverso la descrizione di un esempio. Questo è teso ad illustrare le potenzialità di formalizzazione ed impiego di un modello di natura microeconomica espresso in tutta la generalità, difficilmente ottenibile con strumenti di modellazione tradizionali.

2. MODELLI QUALITATIVI E SIMULAZIONE QUALITATIVA

Diversi sono gli approcci proposti da una branca dell'A.I., la fisica qualitativa, per la modellazione della struttura e del comportamento di sistemi e processi fisici. Tra essi è opportuno segnalare quelli di de Kleer [3] e Forbus [4], ma il punto di vista a cui noi ci riallacciamo è quello di Kuipers [10], [11].

Egli illustra un formalismo atto ad esprimere, in termini qualitativi, una descrizione strutturale che caratterizzi un sistema fisico costituendosi dei seguenti elementi di natura simbolica:

- un insieme $\{f_1..f_m\}$ di simboli di parametri che rappresentano opportuni parametri fisici del sistema variabili su un dato intervallo temporale e derivabili con continuità su di esso
- degli insiemi totalmente ordinati di simboli, associati ciascuno ad un parametro, a rappresentarne particolari valori detti valori landmark
- un insieme di vincoli applicati ai simboli di parametri:
  - ADD(f,g,h) indica che f + g = h
  - MULT(f,g,h) indica che f * g = h
  - MINUS(f,g) indica che -f = g
  - DERIV(f,g) indica che df/dt = g
  - M+(f,g) indica che f cresce monot. al crescere di g
  - M-(f,g) indica che f decresce monot. al crescere di g

Ad ogni vincolo possono essere associate delle corrispondenze di valori tra i parametri da esso interessati. Comecaso particolare si definiscono i due seguenti vincoli con annessa corrispondenza:

- MO+(f,g) indica che f cresce monot. al crescere di g ed, inoltre, che f = 0 quando g = 0
- MO-(f,g) indica che f decresce monot. al crescere di g ed, inoltre, che f = 0 quando g = 0
- un simbolo che rappresenta il punto temporale iniziale t0
- l'insieme dei valori qualitativi iniziali dei parametri fi al tempo t0.

Al riguardo è necessario sottolineare come l'introduzione dei valori landmark consenta di ottenere una sorta di discretizzazione dei ranges relativi ai singoli parametri, ma anche dell'intervallo temporale di simulazione che viene ad essere marcato dagli istanti, punti temporali distinti, in cui un parametro giunge ad avere il valore rappresentato da un landmark.
Lo stato qualitativo di una funzione fi ad un punto temporale distinto o nell'intervallo tra due successivi di essi può quindi definirsi come una coppia $<q_{val},q_{dir}>$ con:

$$q_{dir} = \begin{cases} inc & \text{se } f_i' > 0 \\ std & \text{se } f_i' = 0 \\ dec & \text{se } f_i' < 0 \end{cases}$$

(è nell'intervallo tra due valori landmark)

$$q_{val} = \begin{cases} l_{ij} & \text{se } f_i = l_{ij} \text{ (un valore landmark)} \\ (l_{ij}, l_{ij}+1) & \text{se } f_i \ \Sigma \ (l_{ij}, l_{ij}+1) \end{cases}$$

Volendo, ad esempio, impiegare tale formalismo per modellare il sistema fisico costituito da una massa attaccata ad una molla oscillante su una superficie in assenza di attrito, è possibile allora servirsi di tre simboli di parametri "x", "v" ed "a" ad indicare rispettivamente la posizione, la velocità e l'accelerazione della molla stessa. Per ciascuno di essi dovremo quindi fornire gli insiemi ordinati di valori landmark ed i valori iniziali come indicato nel seguente prospetto (dove "inf" e "minf" denotano rispettivamente $+\infty$ e $-\infty$) :

parametro	valori landmark	q_{val} iniz.	q_{dir} iniz.
x	minf,0,inf	0	inc
v	minf,0,v_init,inf	v_init	std
a	minf,0,inf	0	dec

Per quanto riguarda i vincoli, impiegheremo invece i seguenti:

DERIV(x,v)
DERIV(v,a)
MO-(a,x).

Dalla conoscenza così espressa risulta possibile, tramite un processo descritto dall'algoritmo QSIM, la simulazione qualitativa, derivare una o più possibili descrizioni del comportamento qualitativo del sistema ognuna delle quali è costituita dai seguenti elementi:

- una sequenza {t0..tn} di simboli che rappresentano punti temporali distinti per il comportamento del sistema
- un insieme totalmente ordinato di valori landmark per ciascun parametro. Può essere lo stesso fornito in input all'algoritmo, oppure una sua estensione comprendente nuovi valori, assunti dal parametro in condizioni di derivata nulla, da esso individuati
- una lista di valori qualitativi per ogni parametro che ne esprime il comportamento qualitativo. Tali valori sono relativi alternativamente ad un punto temporale distinto o all'intervallo tra questo ed il successivo.

Nonostante l'algoritmo cerchi di costruire il reale comportamento del sistema fisico, gli accade spesso, a causa di conoscenza non completamente specificata, di vedersi costretto a generare più alternative possibili rappresentate da tutti i cammini che congiungono la radice con le foglie di un albero di stati qualitativi: l'albero di envisioning. Al riguardo è anzi doveroso segnalare come, a causa della località e dell'informazione qualitativa su cui l'algoritmo si basa, sia possibile l'introduzione, a fianco di tutti i comportamenti genuini, anche di comportamenti spurii, impossibili per il reale sistema fisico modellato. La loro eliminazione è uno degli obiettivi più impellenti della ricerca e già iniziano ad ottenersi interessanti risultati al riguardo[12]. Nel frattempo è però prevista dall'algoritmo la gestione di regole euristiche che, formalizzando conoscenza relativa al particolare ambito applicativo, ne favoriscano almeno un tentativo di controllo. Interessanti sono le relazioni, individuate da Kuipers stesso ed espresse in figura 1, tra la descrizione strutturale presentata ed un'equazione differenziale di impiego più comune. Entrambe possono considerarsi utili astrazioni della reale struttura di un sistema così come le funzioni continue, soluzioni dell'equazione differenziale, ed il risultato della simulazione qualitativa modellano entrambe il suo comportamento. Diverso è però il livello di dettaglio espresso dai due formalismi che diversifica le classi di problemi in cui essi sono più proficuamente utilizzabili. E' la comprensione di tale concetto che suggerisce l'impiego dei modelli qualitativi per il supporto decisionale.

3. SIMULAZIONE QUALITATIVA PER IL SUPPORTO DECISIONALE

La prima, e più importante, proposta relativa ad un'estensione delle capacità espressive di QSIM, da cui potrebbe derivare non solo un notevole ampliamento degli ambiti applicativi trattabili ma addirittura lo spunto per una visione del tutto nuova, si concretizzerebbe nell'aggiunta, di scarso impatto sulla struttura dell'algoritmo stesso, della capacità di gestire, come variabile indipendente su cui basare la simulazione, non solo il tempo ma una qualsiasi variabile decisionale indicata dal modello. Si avrebbe quindi il passaggio da un'ottica tesa alla costruzione del comportamento di un sistema al trascorrere del tempo, ad una caratterizzata dalla generazione di tutti i possibili andamenti dei suoi parametri, al variare di una particolare quantità, distinguibili da un punto di vista qualitativo. Una volta definita come variabile decisionale, ad esempio, la quantità di beni da produrre in un dato periodo o il prezzo con cui presentarsi sul mercato e costruito un modello espresso dai vincoli che legano i parametri da questa dipendenti, sarebbe allora possibile demandare all'algoritmo una ricostruzione di tutti i loro possibili andamenti al variare di essa. L'analisi dell'albero così ottenuto, supportata da strumenti opportuni, potrebbe portare, nella mente del decisore, al necessario chiarimento che gli consenta di impostare la strategia più opportuna: di fissare cioè un particolare valore per la variabile decisionale che sortisca determinati obiettivi.

Una doverosa, se pur piccola, variazione della terminologia finora impiegata ci porterebbe pertanto a definire i punti distinti della variabile decisionale con significato analogo a quello proprio dei punti temporali distinti sopra

indicati, sembra inoltre opportuno sottolineare la differenza che viene a comporsi tra lo stato iniziale, l'insieme dei valori dei parametri sull'estremo inferiore dell'intervallo di variabilità della variabile decisionale, e lo stato attuale, l'insieme dei valori dei parametri che rappresentano appunto le attuali condizioni del fenomeno modellato.

Passando alle specifiche per l'ambiente di contorno, parrebbe molto promettente, in fase di definizione del modello, la possibilità di impiegare un editor grafico, che si avvalga quindi di uno schermo appropriato, con capacità di rappresentazione ad alta risoluzione e possibilmente a colori, ed eventualmente della versatilità di uno strumento di puntamento come un mouse. Tramite esso dovrebbe essere molto semplice ed intuitivo indicare la variabile decisionale e definire quindi i parametri ed i vincoli tra essi mediante la costruzione di una rappresentazione, suggerita da Kuipers stesso, in cui ogni parametro potrebbe essere rappresentato da un piccolo nodo da cui si dipartano ed in cui giungano archi orientati che lo connettano con altri nodi dalla forma analoga alle porte logiche usate in microelettronica. Ognuna di queste, rappresentando un vincolo il cui tipo dovrebbe essere caratterizzato dalla sua forma come indicato in figura 2, riceverebbe in ingresso uno o due archi (in base all'arità del vincolo stesso) provenienti dai nodi relativi ad altrettanti parametri e fornirebbe in uscita un arco diretto verso il nodo corrispondente ad un parametro. In figura 3 è mostrata la rete relativa all'esempio della molla precedentemente illustrato.

Per quanto riguarda tali vincoli, già descritti in precedenza, non ci rimane che segnalare la piccola innovazione implicita nell'assunzione che la derivazione, indicata dal vincolo DERIV, sarebbe relativa non al tempo, in generale, ma alla variabile decisionale.
Creato un nodo per un parametro dovrebbe essere molto semplice per l'utente avere accesso ad una finestra in cui indicare il nome del parametro stesso ed i relativi valori landmark.
A questo punto dovrebbe pure essere consentita la specifica di ulteriori vincoli sul parametro come STD(f), INC(f) e DEC(f) ad indicare rispettivamente che f deve rimanere sempre costante, crescente o decrescente nel corso della simulazione. Tale conoscenza, opportunamente sfruttata da QSIM, consentirebbe di modellare l'andamento di molti parametri senza richiedere la rappresentazione delle rispettive derivate, talvolta ingombranti e di nessun interesse.

In modo analogo dovrebbe essere molto semplice indicare eventuali corrispondenze note a priori tra i valori dei parametri coinvolti da un vincolo opportunamente elencati dal sistema. Una parte veramente di notevole importanza nella specifica di un modello è quella relativa alla scelta delle regole di filtraggio globale da attivare nell'ultima fase del blocco la cui iterazione costituisce il processo di simulazione. E' naturale pensare di avere la possibilità di posizionare il cursore su opportune zone del video per attivare o disattivare delle regole standard, è però degna di notevole cura la determinazione della modalità con cui possano essere specificate delle regole di filtraggio aggiuntive per esprimere le euristiche. Esse dovrebbero poter fare riferimento ai valori qualitativi dei parametri ed a relazioni d'ordine tra essi e particolari valori landmark.

La possibilità di esprimerle attraverso le regole di un sistema esperto grazie all'integrazione da noi prospettata in un generatore di D.S.S. intelligente integrato [7], [9] sembra offrire ottime opportunità. Molti infatti hanno cercato di individuare la possibilità di connettere modelli qualitativi con altri più inclini alla rappresentazione numerica. La prospettiva che ci appare più promettente in questo senso consiste in una struttura in cui il sistema esperto possa giovarsi delle funzionalità offerte, ad esempio, da un modello su worksheet opportunamente richiamato o di valori contenuti in un data-base al fine di decidere circa l'eliminazione di un nuovo stato prodotto dalla simulazione o il suo inserimento nell'albero con un'eventuale marcatura utile nella successiva fase di analisi da parte del decisore umano. Caratteristiche generali dell'interfaccia dovrebbero essere la capacità di gestire una pagina virtuale, non limitata allo spazio rappresentato dal video, al fine di favorire la sintesi di modelli anche di dimensioni non banali, la verifica automatica della consistenza del modello, la stampa delle sue caratteristiche, la sua memorizzazione all'interno di un'opportuna biblioteca, il suo richiamo per un'eventuale modifica o per l'attività più interessante: l'interrogazione.

Molto utile sarebbe la possibilità di accludere, al momento della definizione del modello, qualche riga di testo che consenta di documentare, oltre a quanto favorito dalla chiarezza della struttura del modello stesso, lo scopo per cui esso è stato realizzato e le condizioni di applicabilità.
La fase di interrogazione, l'attivazione cioè dell'algoritmo di simulazione sul modello, non può aver luogo se non dopo un'indicazione dei valori qualitativi di ciascun parametro all'estremo inferiore del dominio, al punto distinto iniziale per la variabile decisionale, azione che potrebbe avvenire tramite un'opportuna maschera visualizzata dal sistema.

Il metodo più semplice per l'espressione dei risultati ottenuti consiste nell'indicazione di tutti i possibili andamenti relativi ad uno o più parametri indicati dall'utente.

Sarebbe semplice infatti produrre, per ogni cammino sull'albero e per ognuno di tali parametri, un grafico qualitativo che esprima i valori successivi del parametro stesso rappresentati da punti con ascissa corrispondente ad un particolare punto distinto per la v.d. od a metà tra due tali punti successivi (ad indicare il valore sul corrispondente intervallo) ed ordinata corrispondente ad un valore landmark od a metà tra due valori landmark successivi (ad indicare un qualsiasi valore su tale intervallo). A completare il quadro parrebbe opportuno marcare ciascuno di tali punti con una freccetta che esprima l'andamento crescente, decrescente o costante per il parametro. Anche in questo caso l'esempio mostrato in figura 4 si riferisce ad uno dei possibili comportamenti ricostruiti dalla simulazione qualitativa per il modello della molla.

Per favorire un'analisi più efficace del risultato della simulazione qualitativa, soprattutto nei casi in cui il numero di andamenti alternativi prodotti sia elevato, pare interessante la possibilità di costruire e quindi impiegare, in fase di interrogazione, quella che noi definiamo maschera di stato. Di tali oggetti, espressioni di condizioni sui valori dei parametri, possiamo distinguere due diversi tipi.

Una maschera di stato obbligato esprime condizioni che si ripercuotono su un filtraggio dei cammini sull'albero di envisioning. Di questo vengono infatti a far parte solo i cammini passanti per almeno un nodo che soddisfi tali condizioni.
Una maschera di stato non obbligato esprime invece condizioni che non influenzano la costruzione dell'albero di envisioning ma che caratterizzano stati ritenuti di particolare interesse per il decisore.
Per quanto concerne' q_{val} ' sembra adeguata la possibilità di esprimere relazioni d'ordine rispetto ad un certo valore landmark oppure adeguate congiunzioni (AND), disgiunzioni (OR) o negazioni (NOT) di condizioni così espresse. Relativamente a ' q_{dir} ' è opportuno poter indicare il suo valore (inc, std o dec) o una disgiunzione tra due valori possibili. Ad un oggetto di ciascuna delle classi dovrebbe essere possibile assegnare un nome che lo distingua dagli altri. Questo potrebbe quindi essere impiegato in fase di visualizzazione del grafico qualitativo per marcare i punti in cui la maschera sia soddisfatta.

La possibilità dell'impiego congiunto di più maschere di stato obbligato, teso a determinare tutti gli andamenti tali che, per ognuna, passino per almeno uno stato che la soddisfi, darebbe luogo all'opportunità di ricercare in modo molto semplice gli andamenti ritenuti interessanti. Supponendo, ad esempio, di costruire una maschera di stato obbligato che modelli lo stato attuale del fenomeno ed una che indichi determinati obiettivi che si intendono raggiungere, sarebbe possibile determinare tutti gli andamenti che portano all'obiettivo consentendo di impostare una opportuna strategia consistente in un incremento o decremento della variabile decisionale.

Con una maschera di stato obbligato per la situazione attuale ed una di stato non obbligato per l'obbiettivo si individuerebbero anche andamenti che non portano al successo. Ciò consentirebbe di individuare opportune condizioni sull'andamento dei vari parametri che garantiscano una possibilità di successo.
Altre maschere di stato non obbligato potrebbero evidenziare condizioni particolarmente svantaggiose da cui è bene rifuggire.
L'esempio riportato in seguito è teso a chiarire i concetti qui espressi.

4. UN'IMPLEMENTAZIONE DELL'ALGORITMO DI KUIPERS PER IL SUPPORTO DECISIONALE

Le specifiche appena illustrate hanno trovato parziale realizzazione nel prototipo da noi sviluppato nel linguaggio PROLOG dell'interprete Prolog-2 della Expert Systems International Ltd, su un elaboratore IBM Personal System/2 Mod. 30 (microprocessore 8086) con sistema operativo MS-DOS.
Esso si configura con il motore che, realizzando il processo generate and test descritto da QSIM, necessita di essere affiancato, per un effettivo impiego, dall'editor grafico per la generazione dei modelli (attualmente espressi come data-bases PROLOG) e dal modulo per la visualizzazione dei grafici qualitativi.
Le sue principali caratteristiche comprendono la gestione di:

- una qualsiasi variabile decisionale su cui basare la simulazione
- tutti i vincoli precedentemente descritti
- i vincoli ad arità 1 STD(f), INC(f) e DEC(f)
- cinque regole di filtraggio globale standard attivabili e disattivabili singolarmente:

REGOLA DEL NON CAMBIAMENTO
se, durante la costruzione dell'albero, il figlio di un nodo corrispondente ad un intervallo per la variabile decisionale è uguale al padre viene eliminato
REGOLA DELLA DIVERGENZA
un nodo in cui un parametro diverge (va ad $+\infty$ o $-\infty$) viene considerato nodo terminale, foglia dell'albero di envisioning
REGOLA DEL CICLO
se un figlio è uguale ad un predecessore del padre, viene eliminato e sostituito da un particolare ramo che segnala la ciclicità

REGOLA DELLA QUIESCENZA
un nodo in cui tutti i parametri hanno qdir std è considerato terminale
REGOLA DELLA NON DIVERGENZA
se in un nodo un parametro diverge, il nodo viene eliminato (l'impiego di questa regola disattiva automaticamente la regola della divergenza).
- regole di filtraggio opzionali definibili dal modello (attualmente rappresentate da regole PROLOG)
- maschere di stato obbligato - maschere di stato non obbligato.

5. UN ESEMPIO DI MODELLO QUALITATIVO

Ci addentriamo ora in un esempio, descritto in abbondanza di particolari, relativo alla costruzione di un modello di natura microeconomica che ci consenta di analizzare le possibilità di una sua formalizzazione impiegando gli strumenti appena indicati.
Ulteriori approfondimenti circa le variabili in gioco e le relazioni tra esse possono trovarsi in un qualsiasi testo standard di microeconomia [13], [1], [14].

5.1 Un modello microeconomico

Quello che ci proponiamo è un esperimento per un'analisi delle potenzialità offerte da QSIM nella formalizzazione e l'impiego di un modello, al livello di aggregazione più elevato, della produzione e del mercato relative ad un'azienda manifatturiera.

Indubbiamente le condizioni che via via assumeremo nello sviluppo del modello possono essere considerate abbastanza restrittive delle sue potenzialità d'impiego, esse sono però dettate dalla necessità di contenere il numero dei parametri e delle loro relazioni in un primo esempio e si prestano ad essere facilmente superate con le estensioni da noi stessi suggerite.

Assunta come variabile decisionale la quantità di un certo bene che l'azienda intende produrre e immettere sul mercato in un dato lasso di tempo, una prima assunzione nella nostra trattazione, relativa al mercato, è che esso possa avvicinarsi, almeno entro certi limiti, ad una struttura teorica nota come mercato perfettamente concorrenziale. Esso è caratterizzato dall'esistenza di un considerevole numero di venditori e compratori, rendendo trascurabile il peso economico di ciascuno di essi nella determinazione delle variabili di mercato, dall'omogeneità della merce offerta dai vari venditori, che elimini ogni possibile motivo di preferenza per quello di uno di essi rispetto a quella degli altri, dall'assenza di vincoli di natura istituzionale, economica o finanziaria che impediscano l'ingresso di nuovi venditori sul mercato. Conseguenze immediate di una tale assunzione si esprimono in un'assoluta mancanza di rivalità tra gli offerenti ciascuno dei quali può decidere liberamente delle proprie strategie di vendita prescindendo dal comportamento degli altri. Egli può vendere qualunque quantitativo di merce al presso corrente di mercato, prezzo che da lui non può essere influenzato dato il peso limitato della sua azione sul mercato stesso.

Posta q la nostra variabile decisionale, cioè la quantità di bene che l'azienda produce e vende, abbiamo pertanto, per quanto riguarda i ricavi totali (Rt),

$$Rt = p*q$$

dove p è il prezzo di vendita unitario del prodotto finito, costante e dettato, come assunto dal modello, dal mercato.
Relativamente ai costi totali (Ct) che l'azienda deve sostenere per produrre ed immettere sul mercato tale quantità di bene, si ha invece

$$Ct = CFt + CVt$$

Per la comprensione di tale espressione è necessario sottolineare una importante distinzione che si compone, in microeconomia, tra breve e lungo periodo. Si definisce infatti breve periodo un lasso di tempo nel quale si assume che alcuni fattori produttivi dell'azienda non possano essere modificati nella loro entità a seguito di difficoltà di installazione legate alla natura degli stessi. Ad esso si contrappone il lungo periodo caratterizzato dalla variabilità di tutti i fattori produttivi.

Supponendo che il nostro processo decisionale sia incentrato sul breve periodo, è possibile quindi distinguere tra costi fissi e costi variabili.
I costi fissi (CFt), corrispondenti a quei fattori produttivi per i quali è fissa la quantità disponibile nel breve periodo, sono indipendenti dalla quantità di bene finale prodotta. Tra essi possiamo annoverare gli ammortamenti sui macchinari e gli impianti, gli oneri finanziari sull'immobilizzo di capitali per la loro acquisizione, l'affitto dei locali dell'azienda, le spese generali ed amministrative ecc..
Per costi variabili (CVt) intendiamo invece quelli che mutano al variare della produzione, corrispondenti cioè a quei fattori produttivi per i quali, nel breve periodo, è possibile modificare la quantità utilizzata. Tra essi consideriamo le materie prime ed ausiliarie, la forza motrice ecc..

Assumendo, per semplicità, l'impiego di un solo fattore produttivo variabile, sia v la quantità usata di tale fattore per la produzione della quantità q di prodotto finale. Se c è il costo dell'unità di fattore produttivo, anch'esso costante per l'assunzione di mercato perfettamente concorrenziale, possiamo allora scrivere

$$CVt = c*v$$

Detto ciò passiamo senz'altro a costruire il modello attraverso il quale intendiamo esprimere l'utile per l'azienda in funzione dell'unica variabile su cui, viste le assunzioni di mercato in concorrenza perfetta e di considerazione del solo breve periodo, il decisore è in grado di intervenire: la quantità q.

Sia U l'utile totale per l'azienda nell'intervallo considerato, esso può essere calcolato sottraendo dai ricavi totali i costi totali, quindi

$$\begin{aligned} U &= Rt - Ct \\ &= Rt - (CFt + CVt) \\ &= Rt - CFt - CVt \\ &= p*q - CFt - c*v \end{aligned}$$

Per quanto riguarda la produzione, cioè la relazione tra v e q che ancora ci manca, impieghiamo una modellazione ampiamente descritta in letteratura sotto il nome di funzione di produzione.
Essa è impiegata per esprimere le relazioni intercorrenti tra le quantità di fattori produttivi variabili impiegati e di prodotto finale da esse ottenibile. Nella loro assunzione di impiego di un solo fattore di produzione ci limitiamo a considerare una funzione di una sola variabile per cui possiamo scrivere

$$q = PT(v)$$

E' proprio l'andamento della curva PT che intendiamo esprimere in tutta la generalità possibile attraverso la modellazione qualitativa. Essa non ha generalmente un andamento lineare, per quanto riguarda la teoria della produzione neoclassica, è anzi caratterizzata dalla cosiddetta legge dei rendimenti decrescenti.
Questa può essere chiarita ricorrendo all'impiego della derivata di PT, definiamo quindi il prodotto marginale Pmg come

$$Pmg = \frac{dPT}{dv} = \frac{dq}{dv}$$

Secondo la suddetta teoria, tale funzione deve possedere un caratteristico andamento che la porta a partire dal valore 0 per v = 0, a raggiungere un massimo e quindi a ritornare asintomaticamente a 0 quando v tenda a $+\infty$, ciò comporta per PT un andamento ad S.

Per calcolare U è allora necessario, a questo punto, esprimere v in funzione di q, impiegare cioè l'inverso della funzione di produzione PT. In realtà, noi non ne conosciamo che l'andamento qualitativo della derivata, pertanto passiamo ad un livello di quantità marginali definendo l'utile marginale Umg come:

$$\begin{aligned} Umg &= \frac{dU}{dq} = \\ &= \frac{dRt}{dq} - \frac{dCVt}{dq} - \frac{dCFt}{dq} \\ &= Rmg - Cmg \\ &= p - c*\frac{dv}{dq} \end{aligned}$$

indicando con Rmg il ricavo marginale e con Cmg il costo marginale.
Per la nota proprietà della derivata della funzione inversa possiamo allora scrivere

$$Umg = p - c*1/\frac{dq}{dv}$$

cioé

$$\begin{aligned} Umg &= p - c*1 / Pmg \\ &= p - c*' 1 ' Pmg \end{aligned}$$

dove 1Pmg è definito come l'inverso di Pmg, è quindi un parametro che parte da $+\infty$ per q = 0, scende ad un minimo positivo, e tende poi a $+\infty$ al tendervi di q.

Questa è una semplice equazione differenziale che, ri-

solta, ci consentirebbe di esprimere l'andamento dell'utile in funzione della quantità di bene finale prodotta, ma in realtà noi conosciamo l'andamento di IPmg solamente in termini abbastanza vaghi, impossibili a rappresentarsi attraverso un'equazione. E' proprio qui che ci proponiamo di sperimentare le possibilità offerte da QSIM, e ciò costituisce l'argomento del prossimo paragrafo.

Per quanto riguarda le possibili estensioni al semplice modello qui ottenuto, esse possono, in prima analisi, prevedere l'impiego di più fattori produttivi v1...vn, caratterizzati ciascuno da un corrispondente Pmg, oppure riferirsi a condizioni di mercato meno ideali. Ciò può realizzarsi nell'espressione di opportune relazioni non lineari tra, ad esempio, la quantità di merce venduta ed il ricavo corrispondente, l'impiego cioè di un Rmg non costante.

6. FORMALIZZAZIONE DEL MODELLO

Passando ad esprimere il modello secondo il formalismo proprio di QSIM indichiamo "quant" come variabile decisionale, commentandola, eventualmente, come "quantità di bene finale prodotta e venduta".
Per quanto riguarda i parametri ed i relativi valori landmark possiamo invece rifarci al seguente prospetto:

costo "costo dell'unità di fatt. produttivo" 0, c
inv_prod_marg "inverso del prodotto marginale" 0, inf
d_inv_prod_marg "derivata dell'inverso del prodotto marginale" minf, 0, dipmg_fin
costo_marg "costo marginale" 0, inf
utile "utile totale" minf, costi_fissi, 0, inf
utile_marg "utile marginale" minf, 0, inf
ric_marg "ricavo marginale" 0, p
Le relazioni strutturali tra i parametri sono espresse dai seguenti vincoli rappresentati anche nella rete in figura (in questo caso non esprimiamo corrispondenze):

STD(costo)
STD(ric_marg)
INC(d_inv_prod_marg)
DERIV(inv_prod_marg,d_inv_prod_marg)
DERIV(utile,utile_marg)
MULT(costo,inv_prod_marg,costo_marg)
ADD(costo_marg,utile_marg,ric_marg)

Le condizioni iniziali, corrispondenti a quant = 0, possono essere descritte nel modo seguente:

costo (c, STD)
inv_prod_marg ([0,inf], DEC)
d_inv_prod_marg ([minf,0], INC)
costo_marg ([0,inf], DEC)
utile (costi_fissi, DEC)
utile_marg ([minf,0], INC)
ric_marg (p, STD)

Per modellare "inv_prod_marg" ne introduciamo la sua derivata che, partendo da un valore negativo e rimanendo sempre crescente, giunge, annullandosi, a portare il parametro stesso al suo valore minimo e quindi a crescere. Non esprimendo valori landmark negativi per esso costringiamo inoltre il sistema a costruire solo cammini in cui rimanga sempre positivo.

Esprimiamo la finitezza della capacità produttiva dell'azienda, che non può evidentemente andare all'infinito, ponendo un limite alla crescita di tale derivata che non può superare il valore "dipmg_fin".
Per quanto riguarda l'utile, il valore landmark "costi_fissi" esprime il suo valore iniziale, per quant = 0, corrispondente ai costi fissi totali.

Per la scelta delle regola ed il filtraggio globale, la presenza di un parametro sempre crescente, che non può quindi mai ritornare au un valore precedente, ci autorizza a disattivare la regola del ciclo a favore di un incremento in efficienza dell'esecuzione della simulazione. Attiviamo invece la regola della non divergenza in considerazione del limite che abbiamo imposto alla crescita di "d_inv_prod_marg" e, per lo stesso motivo, aggiungiamo una regola euristica che rende terminale ogni nodo in cui tale parametro vada a "dipmg_fin". L'aggiunta di un'altra regola euristica si rende inoltre necessaria per eliminare tutti i cammini in cui "inv_prod_marg" si annulli.
Ricordando il significato di tale parametro come inverso del prodotto marginale, il suo annullarsi è infatti interpretabile come un divergere all'infinito del prodotto marginale stesso.
Eventualità fisicamente irrealizzabile.

6. 1 Interrogazione del modello

L'attivazione di QSIM sul modello consente di ricostruire i possibili andamenti dei vari parametri, nel nostro caso siamo interessati all'utile, sul dominio che si estende da una produzione nulla fino alla massima capacità produttiva per l'azienda.

Supponendo che essa si trovi in uno stato di pareggio del bilancio, corrispondente ad un utile nullo, ad una produzione che non ha ancora raggiunto il massimo valore per il prodotto marginale, corrispondente al minimo valore per l'inverso del prodotto marginale stesso, è possibile se-

lezionare gli andamenti di nostro interesse con l'inserimento di una opportuna maschera di stato obbligato che indichiamo con il nome "stato_attuale".
Essa è soddisfatta da stati in cui l'utile si annulli mentre l'inverso del prodotto marginale sia decrescente.

A questo punto sarebbe possibile attivare la simulazione qualitativa, ma pare interessante, per favorire la fase di analisi dei risultati, evidenziare gli stati in cui l'utile assuma un massimo relativo positivo. Si assume infatti un incremento dell'utile come obbiettivo per l'azienda stessa. Ciò si realizza con l'inserimento nel modello di un'ulteriore maschera di stato, questa volta non obbligato, soddisfatta da nodi in cui l'utile assuma un qval non negativo ed un qdir std, ed in cui inoltre l'utile marginale sia decrescente, cioè la derivata seconda dell'utile sia negativa.

Sottoponendo finalmente il modello alla nostra implementazione di QSIM si è ottenuto un albero di envisioning esprimente andamenti diversi. Questi potrebbero essere riportati dal sistema con grafici qualitativi simili a quelli visualizzati nelle figure.
Essi si differenziano sostanzialmente solo per il modo in cui giungono al punto di massima capacità produttiva per l'azienda.

Risulta però evidente che la quantità di produzione deve essere aumentata per conseguire l'obiettivo di un incremento dell'utile che l'azienda si è posto. Altrettanto chiaro risulta che tale aumento della produzione non potrà essere indiscriminato. Sempre in dipendenza della capacità produttiva dell'azienda, si potrà infatti andare inontro ad un ottimo relativo, che è anche ottimo assoluto, per l'utile stesso, dopo il quale tale funzione non potrà che diminuire.

7. CONCLUSIONE

Alla luce delle difficoltà intrinseche nel processo decisionale aziendale, e delle grosse attese quindi in merito alla possibilità di realizzare sistemi automatici in grado di fornirgli un adeguato supporto, l'obbiettivo che con questo lavoro si è cercato di raggiungere è la dimostrazione delle potenzialità in tal senso di un nuovo tipo di strumenti di modellazione.

Il software da noi realizzato, che può configurarsi come una sorta di foglio elettronico qualitativo, si presta ad un impiego come sistema a sé stante, ma promette notevoli vantaggi relativamente ad una sua collocazione come modulo in un sistema integrato. OPTRANS, Knowledgeman [Klein 1986 - Holsapple, Tam & Whinston 1987] ed altri oggetti di questo tipo esprimono infatti una nuova interessante filosofia per la realizzazione di sistemi di supporto alla decisione che consente di fornire all'utente un'ampia gamma di strumenti espressivi, in grado di comunicare ciascuno con gli altri, per la definizione di modelli ed il loro impiego negli ambiti più disparati.

Quello che più pare adatto ai modelli qualitativi da noi studiati sembra certamente essere l'alta direzione, il livello gerarchico più elevato del sistema decisionale aziendale. La pianificazione strategica, l'attività che qui si realizza, richiede spesso capacità di formalizzare conoscenza incompleta, ad elevati livelli di astrazione, inadatta quindi a modelli di tipo tradizionale. E' in tali circostanze che appare promettente la possibilità di esprimere l'andamento di una funzione in termini, ad esempio, di valori qualitativi della sua derivata o di parametri ad essa correlati mediante semplici vincoli.

Il nostro lavoro costituisce quindi un primo passo verso la realizzazione di sistemi in grado di esprimere ed impiegare conoscenza espressa sia in forma qualitativa che quantitativa per realizzare un supporto sempre più intelligente al processo decisionale.

8. BIBLIOGRAFIA

[1] Asimakopulos, A., MICROECONOMIA Il Mulino, Bologna 1986.

[2] Bennett, J.L. (ed.), BUILDING DECISION SUPPORT SYSTEMS, Addison-Wesley, Reading, Mass. 1983.

[3] de Kleer, J., QUALITATIVE AND QUANTITATIVE REASONING IN CLASSICAL MECHANICS, in Proceedings of the Fifth International Joint Conference on Artificial Intelligence, Cambridge, Massachusetts, 1977.

[4] Forbus, K.D., QUALITATIVE PROCESS THEORY, in Artificial Intelligence, vol. 24, Elsevier Science Publishers B. V., 1984.

[5] Hollnagel, E., et al. (eds.), INTELLIGENT DECISION SUPPORT IN PROCESS ENVIRONMENT, Springer-Verlag, Berlin 1986.

[6] Holsapple, C.W., Whinston, A.B. (eds.), DECISION SUPPORT SYSTEMS: THEORY AND APPLICATION, Springer-Verlag, Berlin 1987.

[7] Holsapple, C.W., Tam, K.Y., Whinston, A.B., EX-

PERT SYSTEM INTEGRATION", in Silverman, B.G. (ed.), "Expert Systems for Business", Addison-Wesley, Reading, Mass 1987.

[8] Keen, P.G.W., Scott Morton, M.S., DECISION SUPPORT SYSTEMS: AN ORGANIZATIONAL PERSPECTIVE, Addison-Wesley, Reading, Mass 1978.

[9] Klein, M.R., RECENT DEVELOPMENTS IN PC-OPTRANS A KB/DSS GENERATOR, EURO VIII, DSSS and Expert Session, Lisbon 1986.

[10] Kuipers, B. COMMONSENSE REASONING ABOUT CASUALITY: DERIVING BEHAVIOR FROM STRUCTURE, in Artificial Intelligence, vol. 24, Elsevier Science Publishers B.V., 1984.

[11] Kuipers, B. THE LIMITS OF QUALITATIVE SIMULATION, in Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, Los Angeles, CA 1985.

[12] Lee, W.W., Chiu, C., Kuipers, B.J., DEVELOPMENTS TOWARDS CONSTRAINING QUALITATIVE SIMULATION, Department of Computer Science, University of Texas, Austin, Texas, 1987.

[13] Mansfield, E., MICROECONOMIA, Il Mulino, Bologna 1975.

[14] Zamagni, S., ECONOMIA POLITICA. TEORIA DEI PREZZI, DEI MERCATI E DELLA PRODUZIONE, La Nuova Italia Scientifica, Roma 1984.

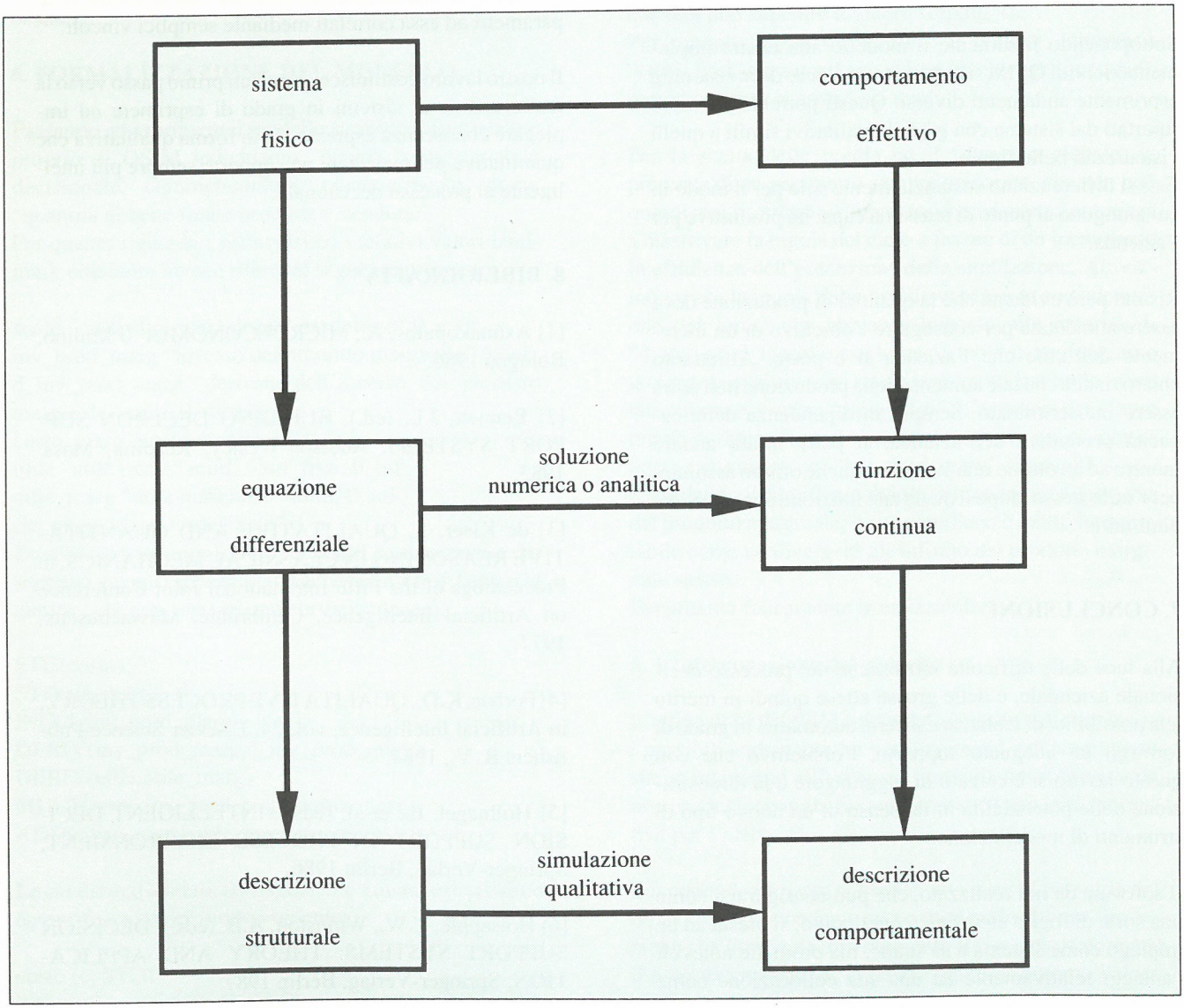


Fig. 1 Equazioni differenziali e simulazione qualitativa come astrazioni del reale comportamento di un sistema fisico.

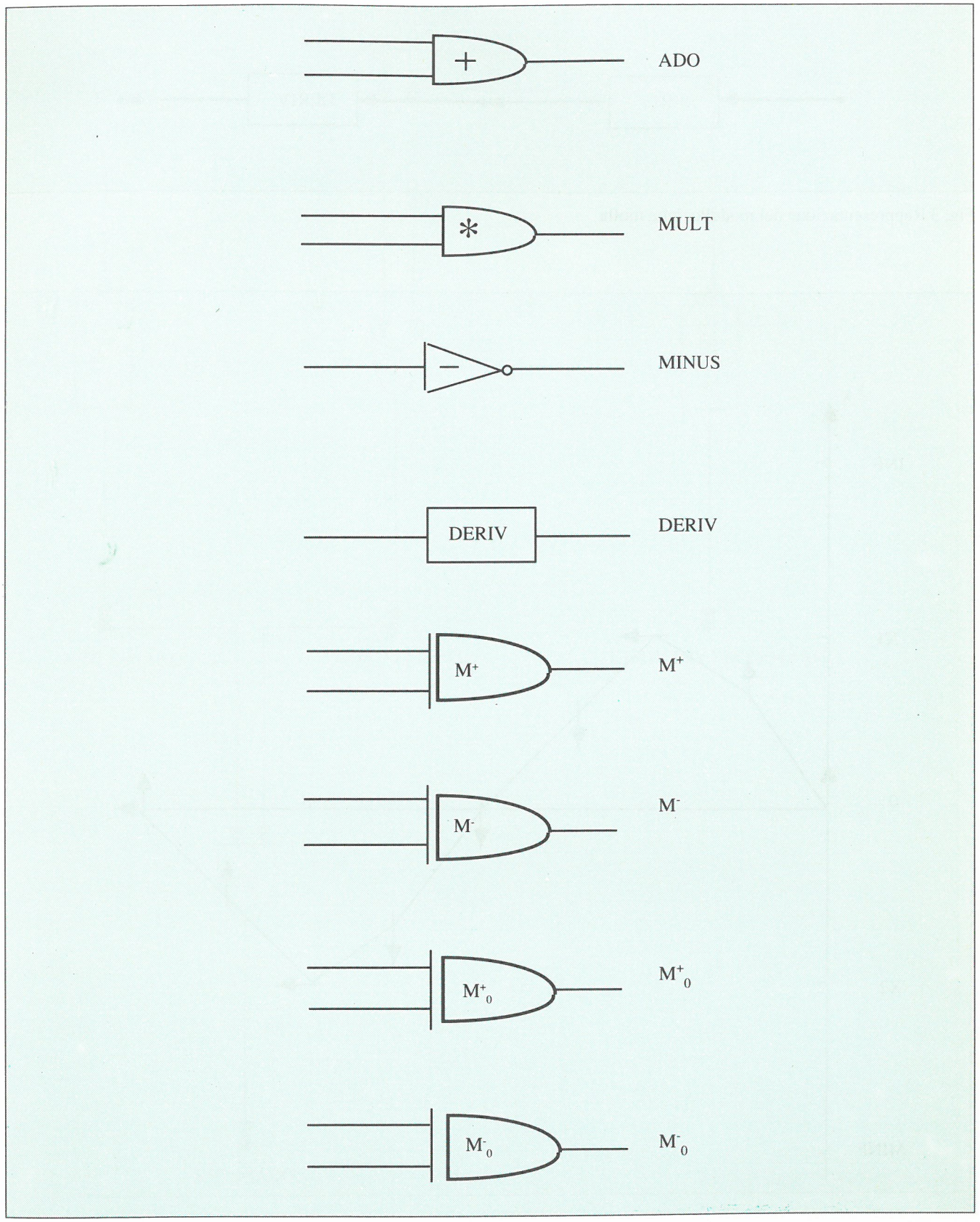


Fig. 2 Possibli rappresentazioni per i vincoli da inserire nei modelli qualitativi impiegabili con qsim.

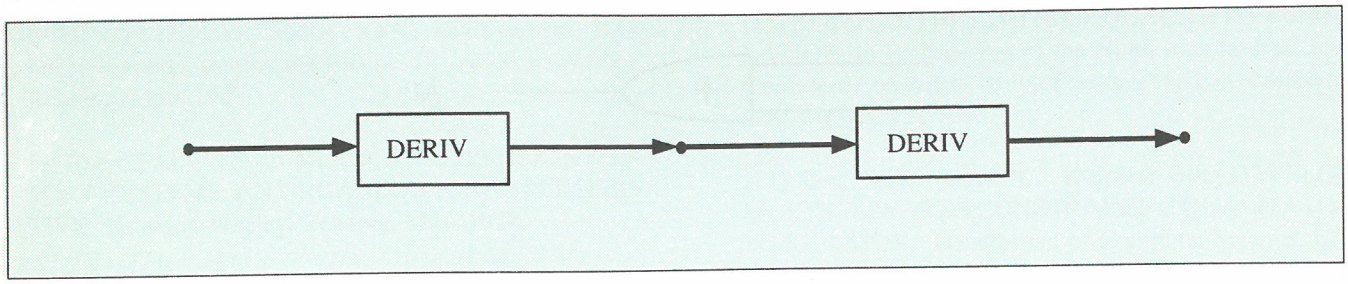


Fig. 3 Rappresentazione del modello della molla

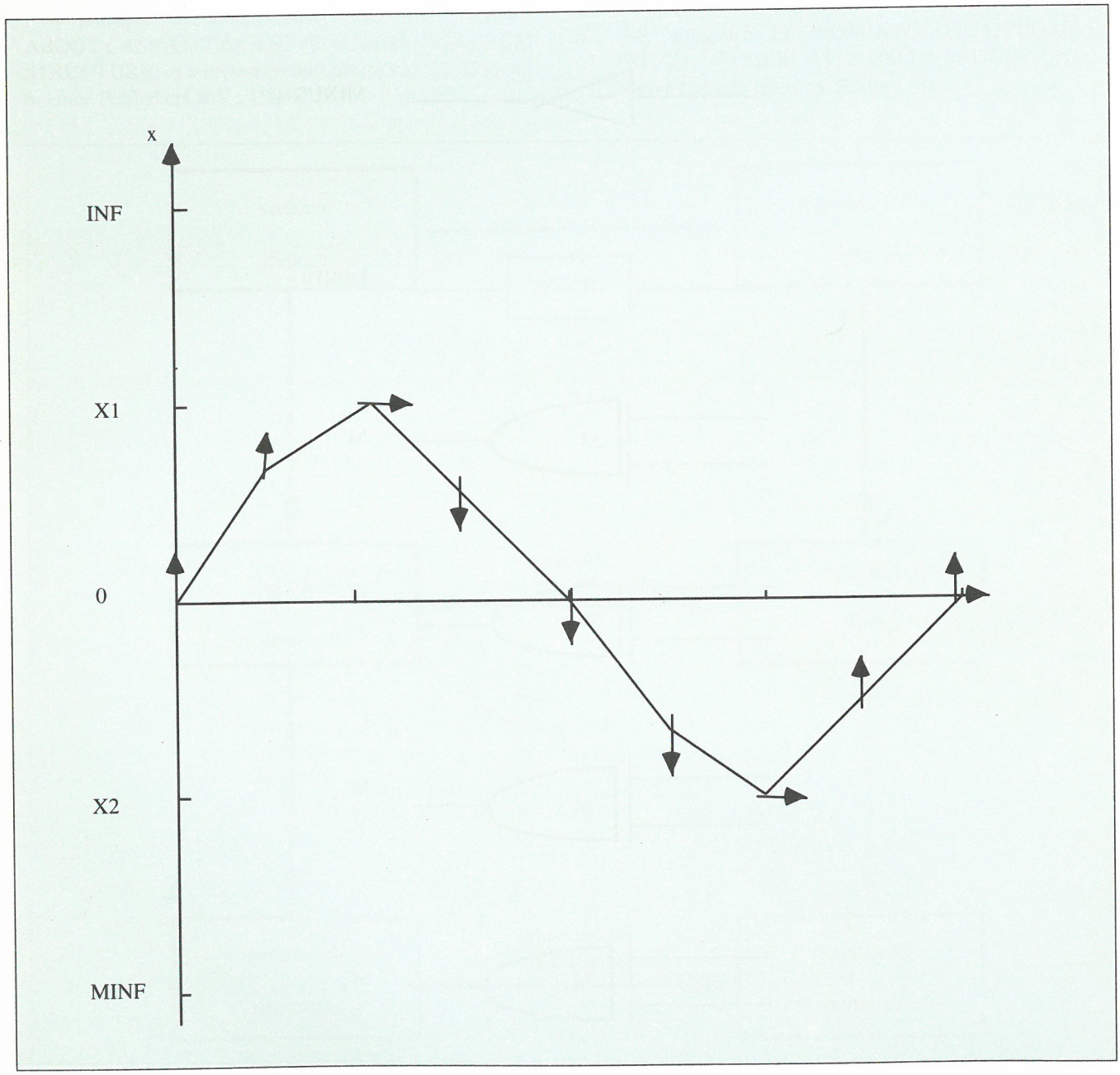


Fig. 4 Grafico qualitativo di uno degli andamenti ricostruiti da qsim per il modello della molla

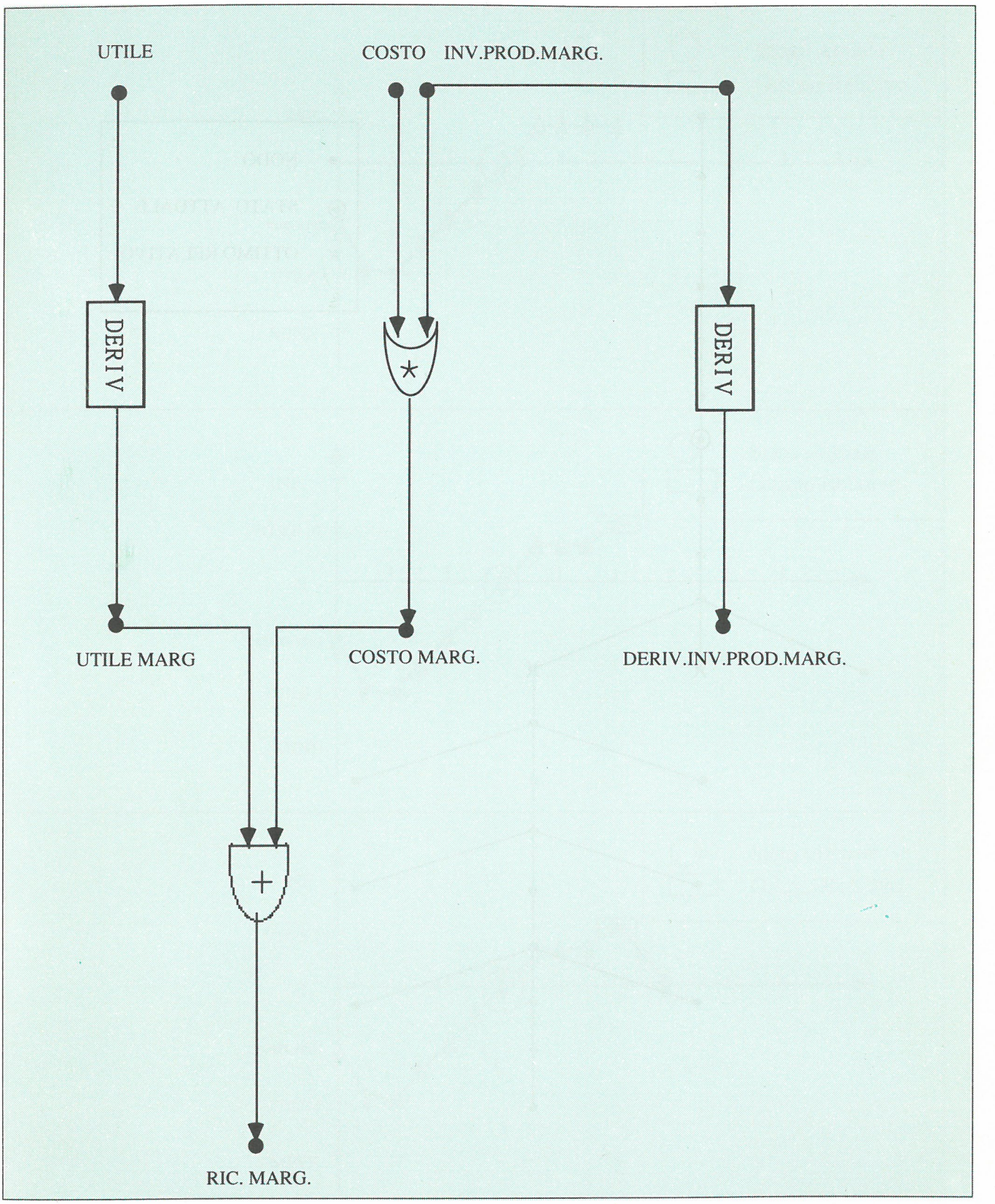


Fig.5 Rappresentazione grafica dei vincoli tra i parametri nel modello dell'esempio. Può essere realizzato con un apposito editor grafico

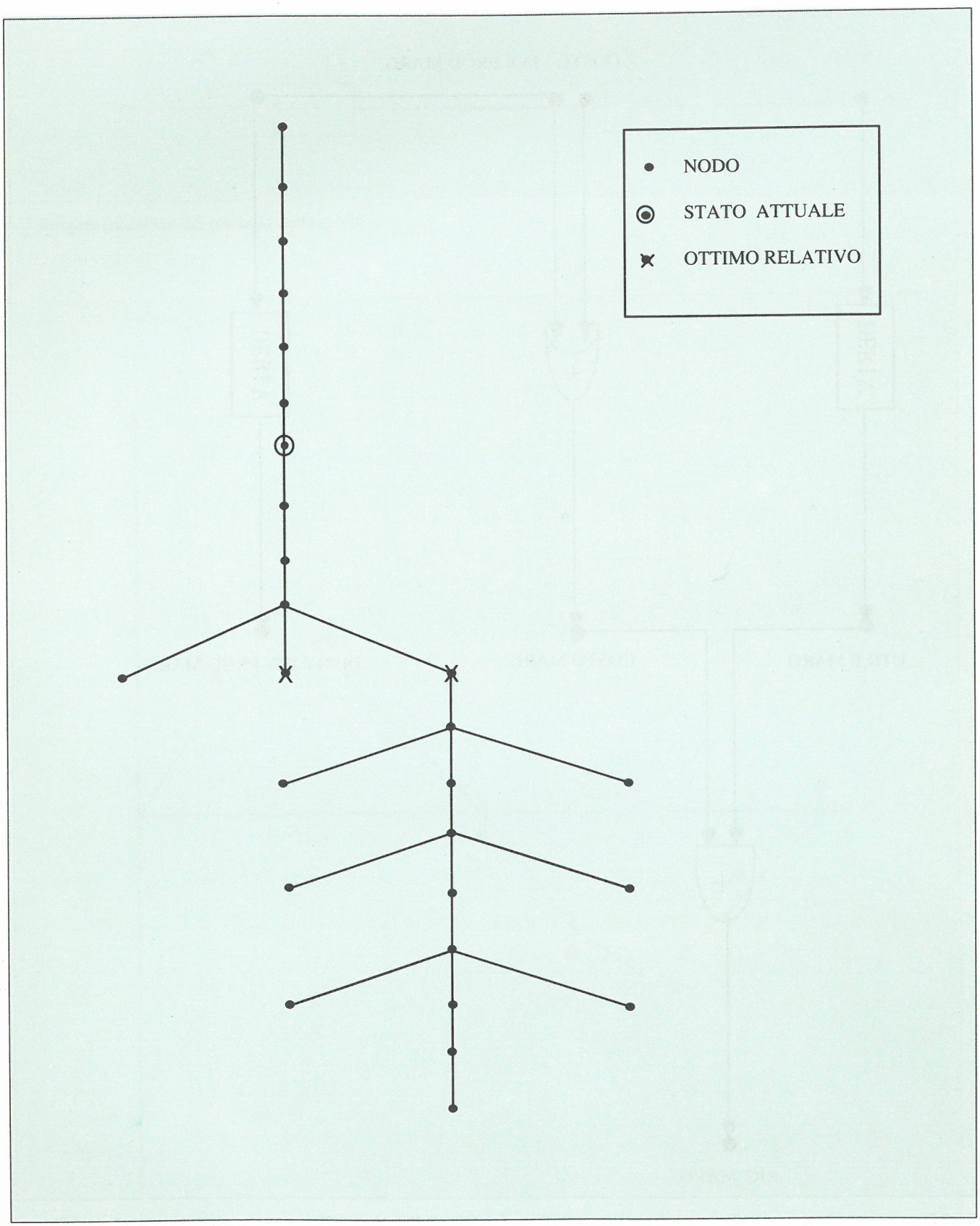


Fig. 6 Albero di envisioning prodotto da qsim sul modello dell'esempio

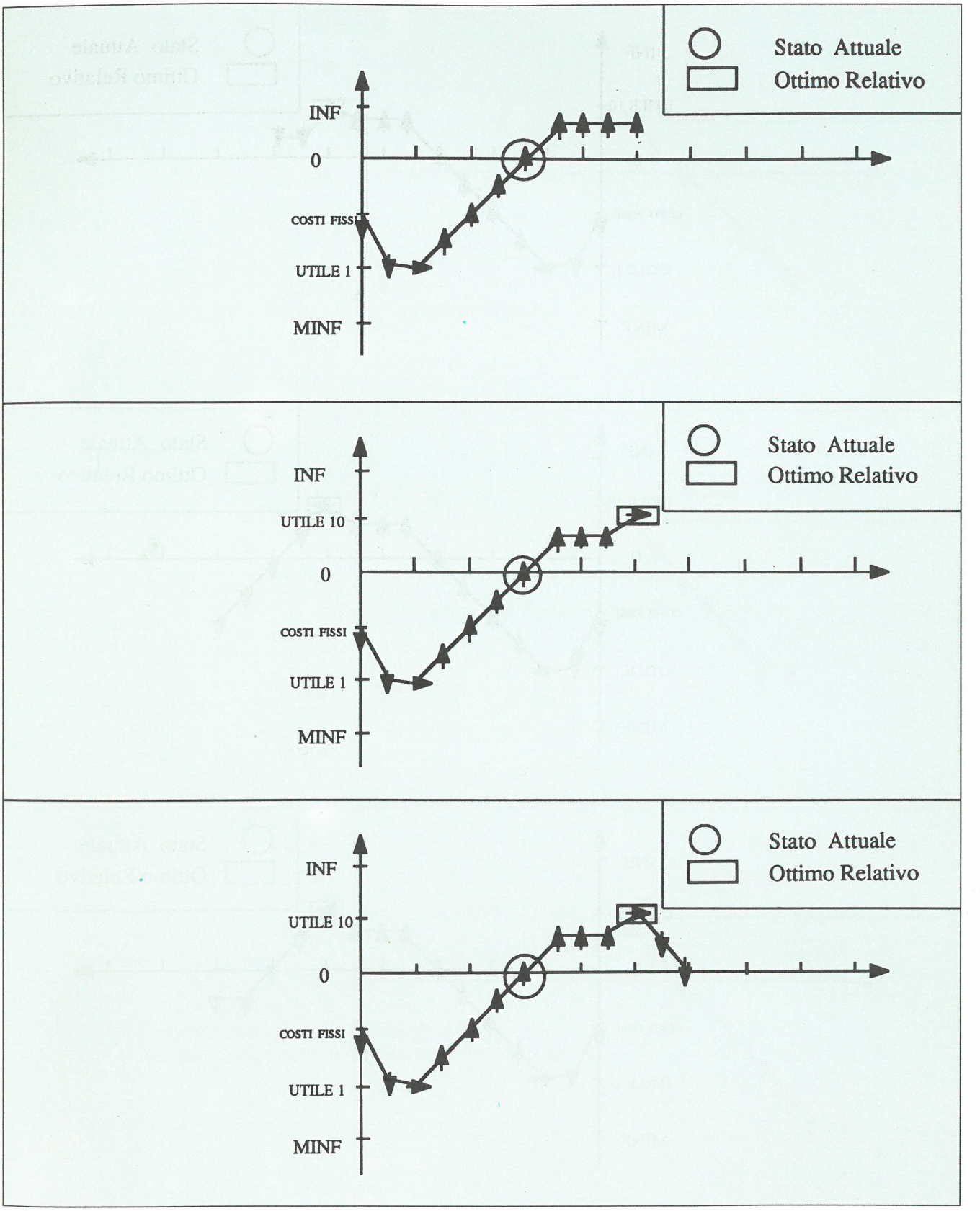


Fig.7 Alcuni andamenti della funzione "utile" prodotti da qsim sull'esempio del modello

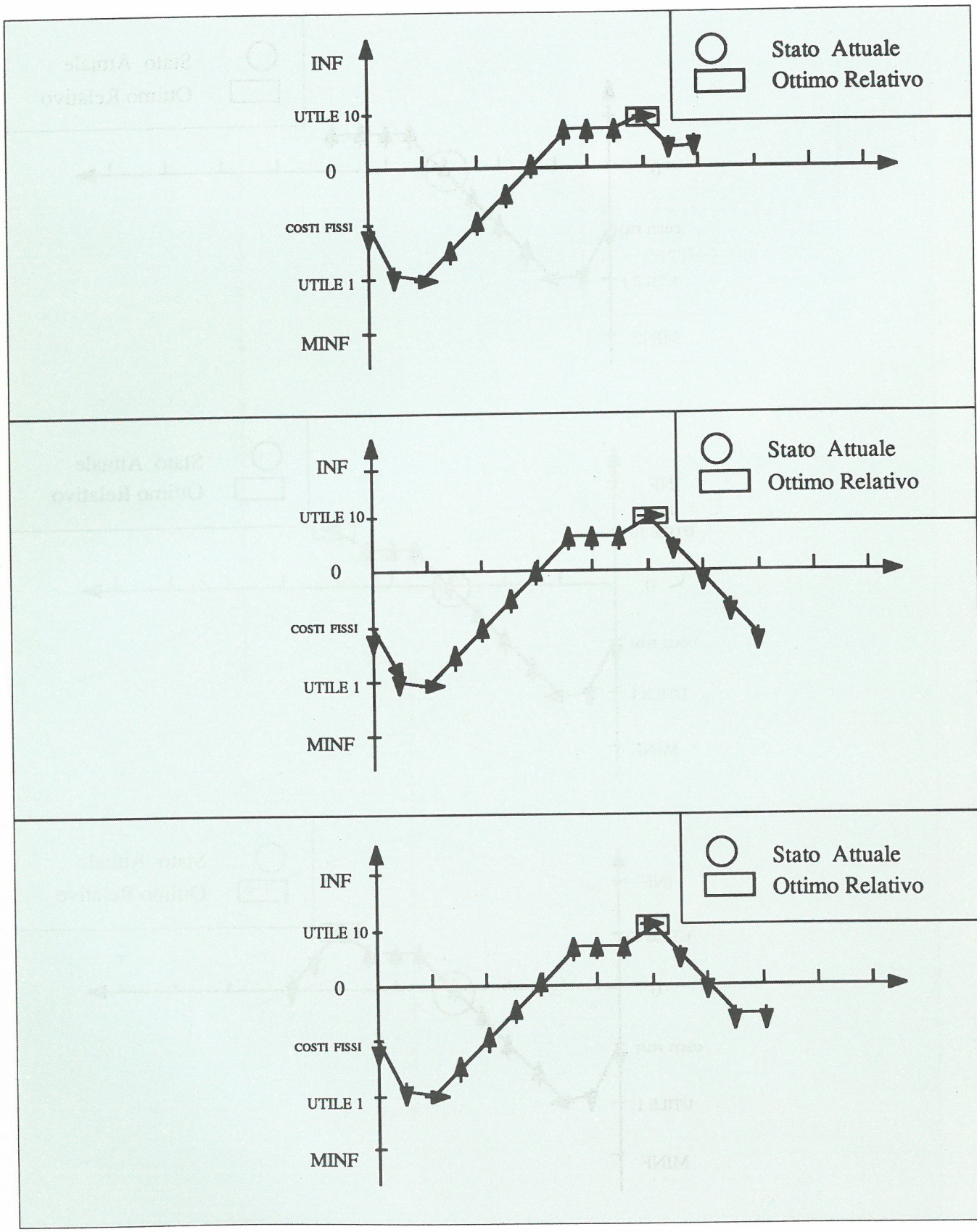


Fig. 8 Altri andamenti della funzione "utile" prodotti da qsim sull'esempio di modello

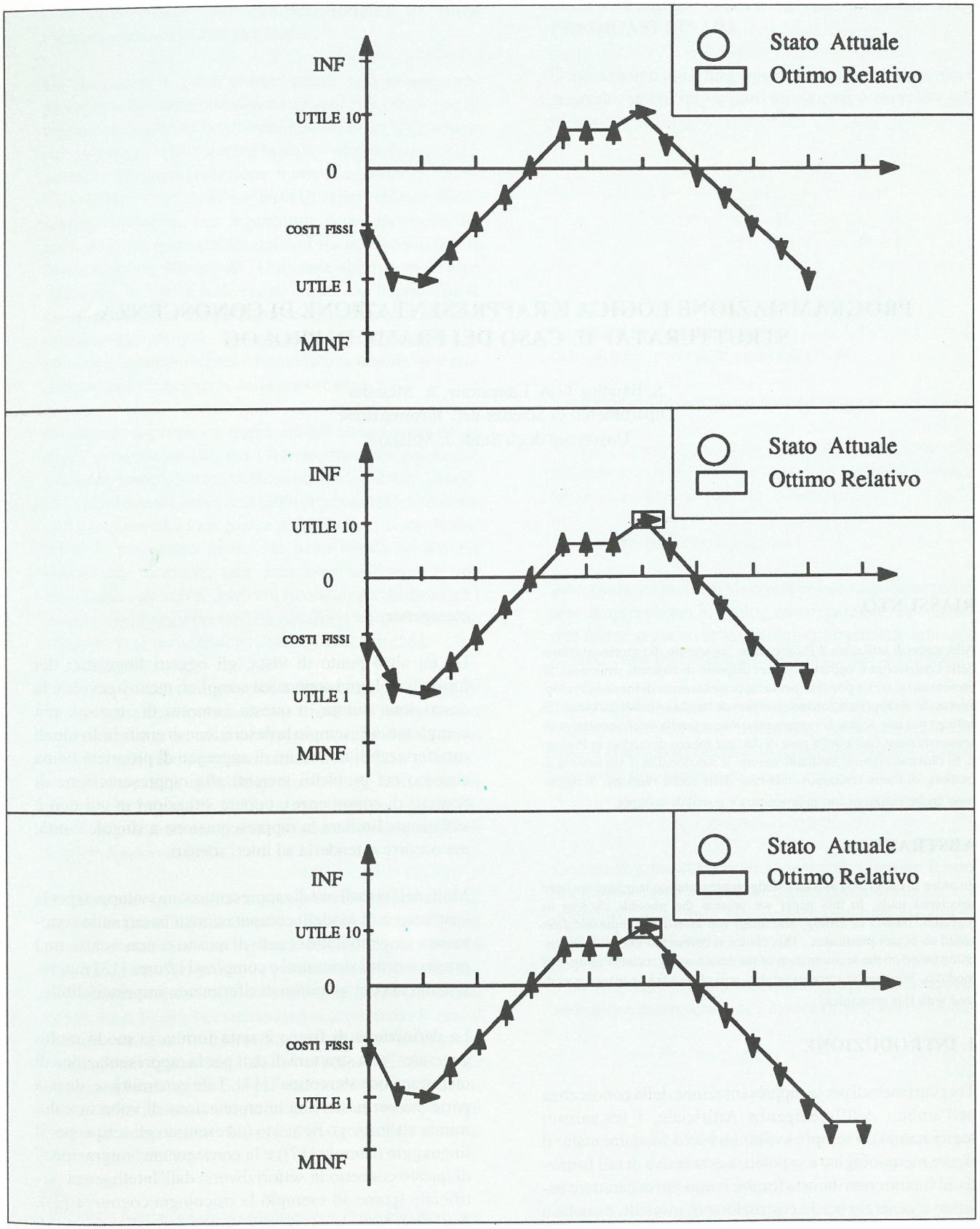


Fig. 9 Ultimi andamenti della funzione "utile" prodotti da qsim sull'esempio di modello

PROGRAMMAZIONE LOGICA E RAPPRESENTAZIONE DI CONOSCENZA STRUTTURATA: IL CASO DEI FRAME IN PROLOG

S. Bandini, G.A. Lanzarone, A. Messina
Dipartimento di Scienze dell' Informazione
Università degli Studi di Milano

RIASSUNTO

Allo scopo di utilizzare il Prolog come linguaggio di rappresentazione della conoscenza è opportuno poter disporre di costrutti strutturati. Si presentano le scelte possibili per la rappresentazione di frame in Prolog, adottando la rappresentazione asserzionale basata su predicati binari. Si collega poi tale scelta di rappresentazione a quella implementativa di segmentazione fisica della base di dati per mezzo di moduli in Prolog-2. Si illustrano quindi mediante esempi le funzionalità di un sistema di gestione di frame realizzato sulla base delle scelte suddette. Si discutono anche relazioni con la letteratura e possibili sviluppi.

ABSTRACT

In order to use Prolog as a knowledge representation language we need structured tools. In this paper we present the possible choices to represent frames in Prolog, and adapt the assertional representation based on binary predicates . This choice is connected to the inplementation based on the segmentation of the data-base by means Prolog-2 of modules. We present some examples regarding a frame manager realized with this principles.

1. INTRODUZIONE

Tra i vari metodi per la rappresentazione della conoscenza nell'ambito dell'Intelligenza Artificiale, i formalismi logici hanno da sempre svolto un ruolo fondamentale. Il rigore metodologico e la potenza espressiva di tali formalismi hanno contribuito a fornire strumenti di carattere basilare e generale per la costruzione di modelli, e anche a definire standard di comunicazione tra ricerche tra loro eterogenee.

Da un altro punto di vista, gli oggetti linguistici dei formalismi logici sono assai semplici; meno agevole è la descrizione diretta in questo contesto di strutture più complesse, ad esempio la descrizione di entità individuali caratterizzabili in termini di aggregati di proprietà (tema classico nei problemi inerenti alla rappresentazione di contesti di conoscenza), oppure situazioni in cui non è sufficiente limitare la rappresentazione a singole entità, ma occorre estenderla ad interi scenari.

Molti dei formalismi di rappresentazione sviluppati per la costruzione di modelli computazionali basati sulla conoscenza tengono infatti conto di questo e, non a caso, tra i primi costrutti strutturati e complessi i *frame* [13] rappresentano ancora un punto di riferimento imprescindibile.

La definizione di frame è stata fornita in modo molto generale: "una struttura di dati per la rappresentazione di una situazione stereotipa" [13]. Tale generalità se, da una parte, ha permesso una interpretazione di volta in volta mirata all'impiego richiesto (ad esempio gli scripts per il linguaggio naturale [17]) e la conseguente "migrazione" di questo concetto in settori diversi dall'Intelligenza Artificiale (come ad esempio la psicologia cognitiva [2]), dall'altra parte la mancanza di una formalizzazione di riferimento lascia aperto il problema della ricostruzione

(sia concettuale che computazionale) di tutta l'impalcatura strutturale dei frame.

Un'eccezione a quest'ultimo punto può essere rappresentata dal fatto che diversi formalismi ibridi per la rappresentazione di conoscenza eterogenea [3], sviluppati soprattutto per i sistemi esperti, comprendono anche strutture di rappresentazione frame-like (come KEE™, Goldworks™, ecc.). L'esigenza di rappresentare conoscenze strutturate, ben organizzate o tassonomiche, ha favorito il recupero dei formalismi frame-like più noti o come supporto alle regole [1] o come alternativa ad esse (pena una forzatura sulle regole stesse con legami che le privano della loro natura frammentaria), permettendo una rappresentazione più consona ai vari tipi di conoscenze coinvolte in un problema e favorendo la nascita di metodologie per l'ingegneria della conoscenza.

Il rapporto tra logica e frame era già stato formulato in modo generale da [9], ma i frame sono stati per lo più utilizzati come costrutti definiti ed implementati ad hoc. Più recentemente, sulla scia della crescente affermazione della programmazione logica e del Prolog, si va rivalutando la possibilità di trovare nella logica un terreno concettuale comune, una notazione uniforme e una tecnologia operativa, entro cui ricondurre e confrontare i diversi "linguaggi speciali" tra i quali appunto i frame, che vengono così recuperati e riformalizzati [8], [18].

Diversi approcci sono presenti in letteratura per la rappresentazione e manipolazione dei frame in Prolog. Essi o fanno ricorso a un metainterprete, usando Prolog come strumento di metaprogrammazione ([4], [18]), o definiscono estensioni dell'interprete Prolog, realizzate ad esempio in Lisp ([10], [14]), o compilano da un formalismo frame-like a Prolog usato ancora come linguaggio di implementazione ([6]).

In questo lavoro si seguirà un approccio che consente di rappresentare e utilizzare frame in Prolog sfruttando il più possibile le caratteristiche proprie del linguaggio e il ruolo dell'interprete. Nel prossimo paragrafo si descrivono e si motivano le scelte di rappresentazione di frame per mezzo di predicati binari; nel successivo si presentano le scelte architetturali per l'implementazione di un sistema per la gestione dei frame secondo la rappresentazione scelta rimandando all'appendice per una descrizione più dettagliata delle funzioni utente attraverso un esempio; nel paragrafo conclusivo si riassumono i risultati e si discutono possibili sviluppi.

™: KEE è marchio registrato dalla Intellicorp.™: Goldworks è marchio registrato dalla GoldHill.

2. RAPPRESENTAZIONE DI FRAME CON PREDICATI BINARI

Consideriamo qui il linguaggio dei frame in un'accezione generale, in cui un oggetto strutturato è espresso nella forma:

$$\begin{array}{c} f \\ \hline s_1 : v_1 \\ \cdots\cdots\cdots \\ s_n : v_n \end{array}$$

dove f è il nome del frame, e per ogni i ($1 \leq i \leq n$), s_i sono detti *slot* e v_i sono detti *valori degli slot*.

Un frame può essere visto sostanzialmente in due modi:

1) come una struttura di dati, cioè un aggregato di informazioni correlate, tale che l'accesso a una di esse consenta un accesso immediato a tutte le altre;

2) come un insieme di proprietà.

A seconda del punto di vista privilegiato, un frame può essere rappresentato in Prolog essenzialmente in una delle due forme seguenti (che costituiscono peraltro una scelta di rappresentazione che si presenta spesso in Prolog [7]):

1) se si guarda il frame come una struttura di dati, poiché oggetti strutturati sono usualmente rappresentati in clausole di Horn come termini, la rappresentazione che ne deriva è del tipo seguente:

$$f(s_1:v_1,\ldots\ldots,s_n:v_n)$$

cioè un termine strutturato avente come funtore il nome del frame e come argomenti coppie (considerando ' : ' un operatore) di termini che definiscono tutti gli slot del frame ed i loro valori ;

2) se lo si guarda come insieme di proprietà, la rappresentazione corrispondente è invece in forma asserzionale:

$$\begin{array}{l} s_1(f,v_1). \\ \cdots\cdots\cdots\cdots \\ s_n(f,v_n). \end{array}$$

dove cioè la base di dati sarà costituita da predicati binari il cui nome è il nome dello slot, il primo argomento è il nome del frame a cui appartiene lo slot ed il secondo argomento è il valore dello slot (la possibilità di rappresentare ogni oggetto strutturato come insieme di relazioni

binarie è mostrata ad esempio in [11] e [15]).

La rappresentazione della gerarchia dei frame richiede nel primo caso un argomento aggiuntivo, che per ogni frame f_1 ne denota il padre f_2; nel secondo caso questo legame può essere espresso da una relazione aggiuntiva, del tipo:

$$is_a(f_1,f_2).$$

Da un confronto tra i due tipi di rappresentazione emergono le seguenti considerazioni. La prima rappresentazione ha come vantaggio la segmentazione del data-base: la conoscenza relativa ad un frame è completamente specificata dagli argomenti del predicato che ha come nome quello del frame, mentre nella seconda la suddivisione logica determinata dal valore del primo argomento dei predicati non è sufficiente a strutturare la base di conoscenza: i predicati binari che rappresentano le informazioni relative ad un frame sono distribuiti in tutto il data-base.

La seconda rappresentazione presenta molteplici vantaggi. Innanzitutto è più consona all'attribuzione di una semantica ai frame (nel modo proposto in [9] e [8]): in un linguaggio logico del primo ordine f e v_i possono essere visti come simboli di costanti e s_i come simboli di funzioni (unarie) o relazioni (binarie); il frame è visto come una proposizione che è soddisfatta da un'interpretazione se e solo se ogni v_i designa lo stesso oggetto di quello ottenuto applicando la funzione o relazione s_i all'oggetto designato da f_i: $<f^i, v_i^i> \in s_i^i (1 \leq i \leq n)$.

La seconda rappresentazione consente inoltre: uniformità dei predicati del data-base: la conoscenza è organizzata in forma di predicati tutti binari, mentre nel primo caso l'arità dei predicati è funzione del numero degli slot che compongono il frame; modificabilità della descrizione: si può modificare con facilità la struttura dei frame variando il numero ed i valori degli slot; le operazioni di aggiornamento sono immediate e non richiedono, come nell'altro caso, di scomporre il predicato principale nei suoi argomenti costituenti per operare poi sulla struttura che rappresenta il singolo slot; massimo sfruttamento del meccanismo di unificazione dell'interprete Prolog, se anche il "frame domanda" (descritto nel seguito) è rappresentato in forma di predicati binari.

Si è scelto di adottare la seconda rappresentazione, per usufruire dei vantaggi che comporta, allo stesso tempo utilizzando un meccanismo di raggruppamento dei predicati di definizione di ogni frame in modo da garantire una organizzazione strutturata della base di conoscenza, in quanto questa è una prerogativa dei sistemi basati sui frame.

Questo approccio permette di rappresentare in Prolog i frame (e in generale strutture complesse) in modo naturale, e di elaborarli usufruendo di tutti i benefici del suo meccanismo inferenziale. Si evita così di usare il Prolog come un semplice linguaggio di programmazione in cui implementare i frame con modalità che finiscono per comportare una perdita sia di dichiaratività che di efficienza, ma allo stesso tempo mantenendo le capacità elaborative caratteristiche dei frame, quali l'ereditarietà.

3. REALIZZAZIONE DEL SISTEMA DI GESTIONE DI FRAME

Una volta fissata la rappresentazione dei frame mediante predicati binari, assunta cioè l'ipotesi ontologica di rappresentare il mondo in termini di individui e loro relazioni, la manipolazione di tali strutture informative costituisce un problema non più di rappresentazione, ma di organizzazione in memoria della rappresentazione, ovvero un problema implementativo [9].

Ciò che è necessario per ottenere i vantaggi discussi prima di segmentazione della base di conoscenza è un sistema Prolog che fornisca la possibilità di definire moduli. Il sistema di gestione dei frame è stato pertanto realizzato [12] usando il Prolog-2 [16].

La segmentazione fisica della base di dati è realizzata associando ad ogni frame un modulo Prolog-2 che ne contiene i predicati di definizione, e sfruttando le primitive di gestione dei moduli messe a disposizione dall'interprete di Prolog-2. Per questo scopo sono stati utilizzati moduli di tipo "data", in quanto ognuno di essi può essere trattato esattamente come se fosse un data-base di clausole: si possono inserire clausole, esaminarle, utilizzarle ed eliminarle. Ogni modulo viene associato al frame del quale contiene i predicati facendone coincidere il nome; anche al file sul quale il modulo verrà memorizzato sarà assegnato il nome del frame.

Seguendo l'architettura presentata si è giunti ad implementare un sistema basato su conoscenza organizzata a frame che comprende un Frame Manager che si occupa della creazione, gestione e presentazione all'utente della base di conoscenza attraverso un'interfaccia di inserimento e costruzione. Essa opera la traduzione dalla sintassi del formato utente del frame alla sua rappresentazione in predicati binari seguendo i passi descritti nell'esempio seguente.

```
FRAME: ALIANTE
is_a: mezzo_aereo
tipo_uso: privato
problemi: maltempo, mancanza_traino
```

secondo la sintassi utilizzata per la rappresentazione (cfr. par. 2) abbiamo:

$f \equiv f_1$= aliante f_2= mezzo aereo
$s_1 \equiv$ is_a $v_1 \equiv f_2$

s_2= tipo_uso v_2= privato
s_3= problemi $v_{3,1}$= maltempo
$v_{3,2}$= mancanza_traino

che dà origine alle asserzioni a predicati binari della rappresentazione finale:

```
is_a (aliante, mezzo_aereo)
tipo_uso (aliante, privato)
problemi (aliante, maltempo)
problemi (aliante, mancanza_traino)
```

Il sistema possiede inoltre un'interfaccia di dialogo con l'utente, che permette l'interrogazione della base di conoscenza.
Le domande che si possono rivolgere ad un Frame System sono espresse mediante la presentazione di una data situazione (descritta anche in modo incompleto con l'utilizzo di variabili) e il sistema ricerca la corrispondenza tra la descrizione fornita e un sottoinsieme dei frame rappresentati nella base di dati. La possibilità di tale "matching" è fornita dalla particolare sintassi che caratterizza ambedue: in questo modo anche una domanda è presentata come un frame. Esempi di costruzione del frame domanda e del funzionamento dell'interfaccia di interrogazione sono forniti in Appendice.
La possibilità di gestire meccanismi di ereditarietà che sono indispensabili per una definizione di frame come mezzi di rappresentazione è affidata alla "definizione" di relazioni transitive tra frame organizzati gerarchicamente. Ci riferiamo a relazioni di tipo `is_a` che, nel nostro sistema, provvedono alla strutturazione gerarchica dei frame: gli attributi che caratterizzano un frame sono quelli espressi dagli slot del frame stesso più quelli definiti dagli slot ereditati dai frame a cui è legato direttamente o indirettamente (transitività) dalle relazioni `is_a`. Un frame può ereditare solo gli slot che non sono già utilizzati esplicitamente per la sua definizione.

La soluzione adottata per l'implementazione consente un veloce reperimento ed aggiornamento delle informazioni e permette di adottare un meccanismo di traduzione immediata del frame dal formato utente ai predicati binari corrispondenti. Nell'utilizzo del Frame System perciò non esistono tempi di attesa dovuti alla compilazione del linguaggio di definizione dei frame dato che questo è stato realizzato direttamente in Prolog; le operazioni che richiedono maggior tempo di esecuzione sono quelle di apertura e salvataggio dei moduli.

In Appendice è riportato un esempio (originariamente scritto in Prolog da S. Costantini) di utilizzo del Frame System che illustra una possibile sessione di lavoro nella quale si esaminano le più significative primitive di gestione dei frame (attraversamento della gerarchia, creazione, cancellazione) e si mostrano le funzionalità del Query Manager. Per motivi di spazio si suppone di disporre già di una base di conoscenza della quale si rappresentano l'organizzazione gerarchica e alcuni frame nel formato utente. La sessione di lavoro è riportata fedelmente anche per quanto riguarda l'interfaccia utente basata su finestre sia per la realizzazione dei menu che per la presentazione dei frame; nell'esempio gli input forniti dall'utente sono quelli scritti in corsivo.

4. CONCLUSIONI E POSSIBILI SVILUPPI

Questo lavoro ha condotto a "rivisitare" la nozione di frame in vista di un suo possibile recupero nell'ambito della programmazione logica. La motivazione delle scelte effettuate nella realizzazione di un sistema di gestione di frame in Prolog costituisce una proposta che mette in luce:

- la possibilità di definire una semantica di riferimento per i frame;
- l'individuazione della rappresentazione più "naturale" in programmazione logica e dell'implementazione più efficiente in Prolog;
- la sostanziale adeguatezza del meccanismo inferenziale basato sulla risoluzione a trattare aspetti tipici dei frame come l'ereditarietà.

La principale caratteristica non trattata è quella relativa al ragionamento per default. Come discusso in [9], questo sembra essere l'unico aspetto dei frame non riconducibile al linguaggio e all'apparato deduttivo della logica ordinaria, in quanto richiede un ragionamento riflessivo, ovvero la capacità del sistema inferenziale di utilizzare dinamicamente una rappresentazione di se stesso. Una possibilità di continuazione del lavoro in questa direzione è correlata a una ricerca in corso sull'uso della riflessione in ambito logico [5].
In questo contesto è possibile inquadrare anche due altri aspetti passibili di approfondimento. Il primo è che, per un reale confronto di efficienza tra due programmi funzionalmente equivalenti (quale sarebbe necessario per giustificare quantitativamente l'affermazione sulla maggiore

efficienza ottenuta con la rappresentazione a predicati binari) occorrerebbe definire un'unità di misura standard (ad esempio, numero di unificazioni nell'esecuzione di un goal) e poterla poi effettivamente misurare; un sistema inferenziale che ha una rappresentazione di se stesso si presta ad ottenere tali rilevazioni [5].

Un secondo aspetto attiene a una metodologia di uso di frame in ambito logico. In un sistema riflessivo si può pensare di passare mediante riflessione dall'inferenza usuale a livello oggetto a un'inferenza per default a metalivello, solo quando questa è effettivamente utilizzata; si eviterebbe così, anche con questa estensione, il ricorso generalizzato a un metainterprete quale quello di [4]. Questo costituirebbe un esempio di quali sono gli usi possibili della riflessione, questione che è attualmente un problema aperto (cfr. [8], p.261).

RIFERIMENTI

[1] Aikins J.S., A RAPRESENTATION SCHEME USING BOTH FRAMES AND RULES, in B.G. Buchanan e E.H. Shortliffe (eds), "Rule-Based Expert Systems", Addison Wsley, Readings 1984

[2] Airenti G., Bara B.G.,Colombetti M., RAPPRESENTAZIONE ED USO DELLA CONOSCENZA NELLA SIMULAZIONE COGNITIVA, Ricerche di Psicologia, III,II-12, 1979

[3] Bandini S.,Spampinato L., REPRESENTING HETEROGENEUS KNOWLEDGE IN EXPERT SYSTEMS: AN APPLICATION FOR TOXICOLOGY, Procs. 7th International Congress of Medicine and Informatics, MIE 87, Roma, 1987

[4] Bowen K.A., META-LEVEL PROGRAMMING AND KNOWLEDGE REPRESENTATION, New Generation Computing, 3, 1985

[5] Costantini S. , Lanzarone G.A., UN 'ARCHITETTURA RIFLESSIVA PER I LINGUAGGI LOGICI, 3° Congresso Nazionale del Gruppo Ricercatori e Utenti di Logic Programming, Roma, 1988

[6] ESP FRAME ENGINE, User Manual, Expert Systems International Ltd., 1986

[7] Furlan F., Lanzarone G.A., PROLOG, LINGUAGGIO E METOLOGIA DI PROGRAMMAZIONE LOGICA, F. Angeli ed., Milano, 1988

[8] Genesereth M.R., Nilsson N.J.,LOGICAL FOUNDATIONS OF ARTIFICIAL INTELLIGENCE, Morgan Kaufmann Publishers Inc., Los Altos, 1987

[9] Hayes P.J.,THE LOGIC OF FRAMES, in D. Metzing (ed.), "Frame Conceptions and Text Understanding, de Gruyter, 1979

[10] Ito H., Ueno H., ZERO: FRAME + PROLOG, in Wada (ed.), Procs 4th Conference on Logic Programming, Tokyo, July 1985, Lecture Notes in Computer Science n. 221, Springer-Verlag, 1985

[11] Kowalsky R., LOGICS FOR PROBLEM SOLVING, North-Holland Elsevier, New York, 1979

[12] Messina A., PROLOG COME LINGUAGGIO PER SISTEMI BASATI SULLA CONOSCENZA: VALUTAZIONI ED ESTENSIONI, Tesi di laurea in Scienze dell'Informazione, Università degli Studi di Milano, a.a. 1986-1987

[13] Minsky M., A FRAMEWORK FOR REPRESENTING KNOWLED, in Winston P. (ed.), "The Psychology of Computer Vision", McGraw Hill, New York, 1975

[14] Nakashima H., KNOWLEDGE REPRESENTATION IN PROLOG/KR, in International Symposium on Logic Programming, Atlantic City, February 6-9 1984,

[15] Nilsson N.J., PRINCIPLES OF ARTIFICIAL INTELLIGENCE, Tioga Publishing Co., Palo Alto, 1980

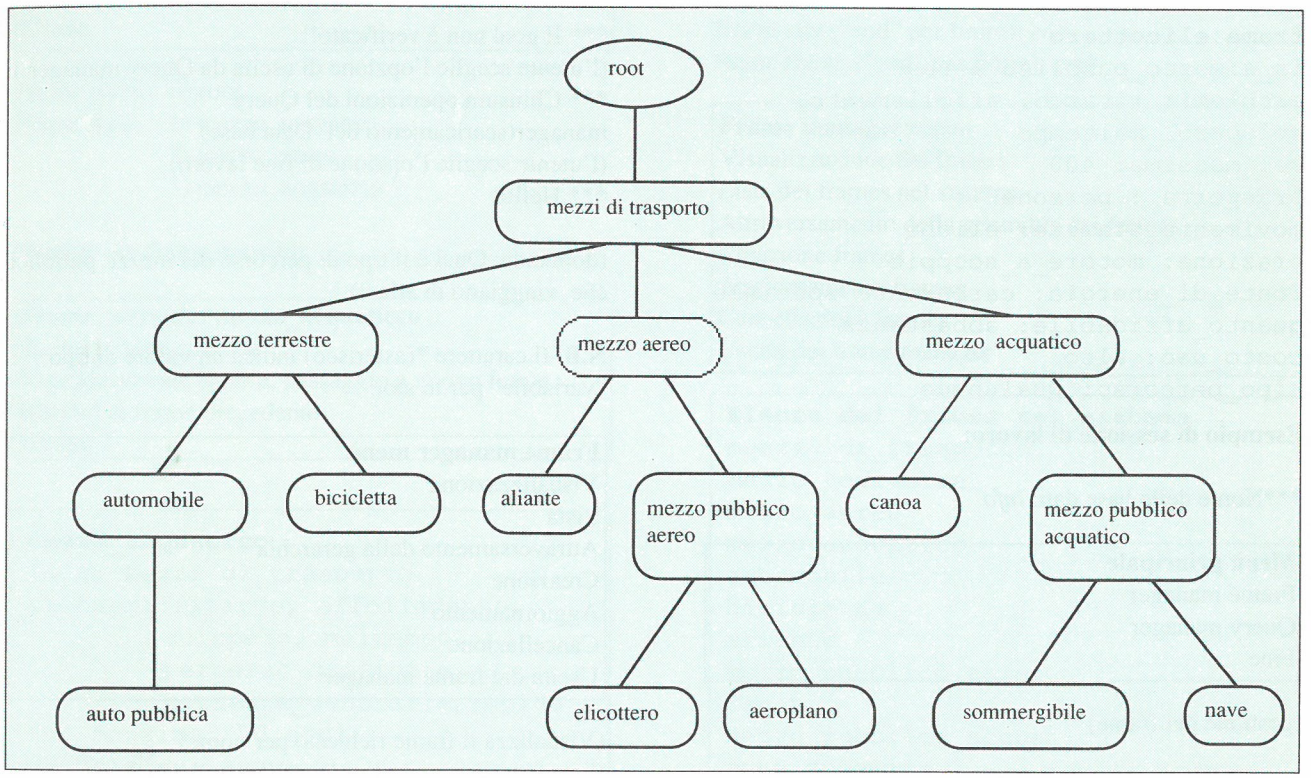
[16] PROLOG-2 Reference Manual Expert Systems International Ltd., Oxford, 1986

[17] Schank R.C., Abelson R.P., SCRIPTS, PLANS, GOALS AND UNDERSTANDINGS, Lawrence Erlbaum, Hillsdale, New Jersey, 1977

[18] Walker A., McCord M., Sowa J.F., Wilson W.G., KNOWLEDGE SYSTEMS AND PROLOG - A LOGICAL APPROACH TO EXPERT SYSTEMS AND NATURAL LANGUACE PROCESSING, Addison-Wesley, Reading, 1987

APPENDICE

Per illustrare le possibilità offerte dal sistema, viene mostrato un esempio di sessione di lavoro nella quale sono utilizzate le primitive di gestione frame e le opzioni di ricerca delle informazioni. Come punto di partenza supponiamo di disporre di una base di conoscenza organizzata in modo tale da classificare alcuni mezzi di trasporto in base alla gerarchia tra gli oggetti ed alle informazioni che li definiscono, come di seguito rappresentato:



la gerarchia dei frame

frame :mezzi _di _trasporto
is_a : root
tipo_trasporto :terrestre, aereo, acquatico
tipo_uso :privato, pubblico
trasporta:persone, cose

frame mezzo_pubblico_aereo
is_a:mezzo_aereo
tipo_uso :pubblico
problemi:sciopero, ritardo
quanto_affidabile: molto

frame :mezzo_aereo
is_a:mezzo_di_trasporto
problemi:ritardo,
affollamento,sciopero, maltempo,
dirottamenti, mancanza_traino
vel_massima: velocità_del_suono
movimento_tramite: ali, eliche
trazione: motore_a_reazione,
motore_a_scoppio,
correnti_aeree
fonte_di_energia:
carburante_speciale,
energia_eolica
viaggia_in: aria

frame:aliante
is_a: mezzo_aereo
tipo_uso: privato
problemi: maltempo, mancanza_traino
vel_massima: 160
trasporta: persone
movimento_tramite: ali
trazione: correnti_aeree
fonte_di_energia: energia_eolica
quanto_affidabile: molto
costo_uso: medio
tipo_percorso:
condizionato_dalle_correnti

frame:aeroplano
is_a:mezzo_pubblico_aereo
problemi: sciopero, ritardo; affol-

```
lamento, maltempo,
dirottamento
vel_massima: velocità_del_suono
movimento_tramite: ali
trazione: motore_a_reazione
fonte_di_energia: carburante_speciale
quanto_affidabile: abbastanza
costo_uso: molto_alto
tipo_percorso: prefissato
```

frame:elicottero

```
is_a:mezzo_pubblico_aereo
problemi: ritardo, affollamento,
sciopero, maltempo
vel_massima: 400
trasporta : persone
movimento_tramite: eliche
trazione: motore_a_scoppio
fonte_di_energia: carburante_speciale
quanto_affidabile: abbastanza
costo_uso: alto
tipo_percorso: qualunque
```

Esempio di sessione di lavoro:

***Nome della base dati: *info*

Menù principale
Frame manager
Query manager
Fine

(gestione dei frame)

Nome dello slot("end" per finire) : *tipo_percorso*
Vaalore dello slot**tipo _percorso** ("end per finire")
:*
Nome dello slot ("end" per finire): *tipo _uso*
Valore dello slot ** tipo _uso ** ("end" per finire) : *pubblico*
Valore dello slot **tipo_uso** ("end" per finire) : *end*
Nome dello slot ("end per finire): *viaggia_in*
Valore dello slot** viaggia_in** ("end" per finire):*aria*
Valore dello slot** viaggia_ in** ("end" per finire):*end*
Nome dello slot ("end" per finire): *end*

*** Il gol è verificato da frame <<aeroplano>>
tipo _percorso:prefissato
tipo _uso:pubblico
viaggia _in: aria

Ricerca di altre soluzioni?
Ok
Cancel

*** Il gol è verificato dal frame <<elicottero>>
tipo _percorso:qualunque
tipo _uso:pubblico
viaggia_in: aria

Ricerca di altre soluzioni?
Ok
Cancel

*** Il goal non è verificato!!
(l'utente sceglie l'opzione di uscita da Query manager)
*** Chiusura operazioni del Query
manager(scaricamento del Data base)
(l'utente sceglie l'opzione di fine lavoro)
*** Hello!

(domanda: Qual è il tipo di percorso dei mezzi pubblici che viaggiano in aria ?)

N.B. Il carattere *(asterisco) indica un valore di tipo "variabile" per lo slot.

Frame manager menù
Visualizzazione
Lista
Attraversamento della gerarchia
Creazione
Aggiornamento
Cancellazione
Uscita dal frame manager

(Visualizza il frame richiesto per nome)
(lista frame secondo l'ordine di creazione e li visualizza a richiesta)
(consente di pecorrere la gerarchia partendo da un punto qualsiasi sc)
(permette di creare nuovi frame , inserendoli nella gerarchia)
(permette di aggiornare gli slots)
(Consente di cancellare i frame o di rimuoverli solamente dalla struttura gerarchica conservando le informazioni che li caratterizzano

Nome del frame corrente:mezzo_aereo

Frame corrente :mezzo_aereo
Stampa frame corrente
Visualizzazione legami, passaggio a nuovo frame
Ritorno al frame precedente
Uscita

Frame padre: 0) mezzi_ di_trasporto
Frame figli: 1) aliante
2) mezzo_pubblico_aereo

Numero del frame corrente: *0*

Frame corrente:mezzi_di_trasporto
Stampa frame corrente
Visualizzazione legami, passaggio a nuovo frame
Ritorno al frame precedente
Uscita

Frame padre: 0) root
Frame figli: 1) mezzo_terrestre
2) mezzo_aereo
3) mezzo_acquatico

Numero del frame corrente: 3

Frame corrente: mezzo_acquaticco
Stampe frame corrente
Visualizzazione legami, passaggio a nuovo frame
Ritorno al frame precedente
Uscita

```
mezzo_acquatico
is_a: mezzi_di_trasporto
problemi:ritardo, affollamento,
         sciopero, maltempo,
         percorso_difficile,
         corrente_contraria,rapide
vel_massima: 50_nodi
viaggia_su: acqua
movimento_tramite:eliche,pagaia
trazione: motore_a _scoppio,
           persona_che_usa_il _mezzo
fonte_di_energia:
nafta,energia_muscolare
```

(l'utente sceglie l'opzione di uscita)

Frame manager menu
Visualizzazione dei frames
Lista dei frames nel sistema
Attraversamento della gerarchia dei frames
Creazione frames
Aggiornamento frames
Cancellazione frames
Uscita dal frame manager

Nome frame ("end" per finire): *taxi*
Valore dello slot "is_a": *auto_pubblica*
Nome slot ("end"per finire): *problemi*
Valore dello slot ** problemi ** ("end" per finire) : *sciopero*
valore dello slot ** problemi ** ("end" per finire) : *traffico*
Valore dello slot ** problemi ** ("end"per finire) : *end*
Nome slot ("end" per finire) : *costo_uso*
Valore dello slot ** costo_uso ** ("end" per finire) : *alto*
Valore dello slot ** costo_uso ** ("end" per finire): *end*
Nome slot ("end" per finire): *end*
Nome frame ("end" per finire): *end*

Frame manager menu
Visualizzazione dei frames
Lista dei frames nel sistema
Attraversamento della gerarchia dei frames
Creazione frames
Aggiornamento frames
Cancellazione frames
Uscita dal frame manager

```
Elenco dei frames nel sistema
mezzi _di_trasporto
mezzo_terrestre
mezzo_aereo
mezzo_acquatico
automobile
bicicletta
aliante
mezzo_pubblico_aereo
canoa
mezzo_pubblico_acquatico
auto_pubblica
elicottero
aeroplano
sommergibiile
nave
taxi
```

Nome del frame da cancellare ("end" per finire): *mezzo_pubblico_acquatico*

Cancellazione
cancellazione
Backup del frame

Menu principale
Frame manager
Query manager
Fine

(permette la costruzione di un "frame domanda"e ne cerca tutti i "match" tra i frame della base di conoscenza)

*** Caricamento del Data base

Nome dello slot ("end" per finire): *viaggia _in*
Valore dello slot ** ("end" per finire):*aria*
Valore dello slot ** (" end" per finire) : *end*
Nome dello slot ("end" per finire): *tipo_ percorso*
Valore dello slot ** tipo_percorso** ("end" per finire): *prefissato*
Valore dello slot ** tipo_ percorso ** ("end" per finire): *end*
Nome dello slot ("end" per finire): *problemi*
Valore dello slot ** problemi ** ("end" per finire): *dirottamenti*
Valore dello slot ** problemi ** ("end" per finire): *sciopero*
Valore dello slot ** problemi ** ("end" per finire): *end*
Nome dello slot("end" per finire): *end*

*** Il goal è verificato dal frame <<aeroplano>>
viaggia_in : aria
tipo_percorso: prefissato
problemi: dirottamenti, sciopero, ritardo, affollamento, maltempo

(domanda: quali sono i mezzi che viaggiano in aria, seguendo un percorso prefissato ed hanno il problema dei dirottamenti e dello sciopero?)

Ricerca di altre soluzioni?
Ok
Cancel

*** Il goal non è verificato!!

Altra richiesta?
Ok
Cancel

LINGUAGGI OBJECT-ORIENTED: IMPLEMENTAZIONE DI UN SISTEMA A FLAVORS IN LISP

Danilo Dabben
Laboratorio di Intelligenza Artificiale
CSI Piemonte - Torino

RIASSUNTO

Flavor System è un ambiente per la programmazione object oriented attualmente disponibile su Lisp Machines ma non ancora supportato nel Common Lisp Standard. L'articolo descrive un'implementazione di queste primitive (su PC 286/386) che potenzia ed arricchisce l'ambiente Golden Common Lisp.

ABSTRACT

The Flavor System is an Object Oriented Programming Environment currently avaliable on Lisp Machines, but not supported in standard Common Lisp yet. This paper describe an implementation of such primitives, avaliable on PC 286/386, making richer and more powerful Golden Common Lisp environment.

1. NOTE SUI SISTEMI DI PROGRAMMAZIONE OBJECT-ORIENTED

La programmazione object-oriented deriva dal concetto di "tipo di dati astratti" applicato, prima che nell'Intelligenza Artificiale, in linguaggi per la simulazione (SIMULA) o ispirati al Pascal ma volti alla costruzione di sistemi complessi (ADA).

Il concetto base è quello di oggetto: una unità che associa dei dati e delle operazioni su di questi. I dati contenuti nell'oggetto non sono visibili direttamente dagli utenti ma solo utilizzabili tramite le operazioni definite su di essi. E' evidente l'ispirazione fornita dai sistemi operativi: gli oggetti sono analoghi ai processi di un S.O. e comunicano fra loro (come i processi) tramite scambio di messaggi.

I principali linguaggi per la programmazione object-oriented normalmente utilizzati nelle ricerche di Intelligenza Artificiale sono attualmente SMALLTALK, INTERLISP/LOOPS, ZETALISP/FLAVORS (SMALLTALK è un "vero" linguaggio object-oriented uniforme, dove cioè esistono solo oggetti e scambi di messaggi, LOOPS e FLAVORS sono invece degli ambienti Lisp accresciuti con delle primitive object oriented, ma dove comunque è utilizzata la programmazione Lisp). Possiamo inoltre considerare anche il formalismo ad attori che, pur non essendo un linguaggio vero e proprio, è comunque un modello di programmazione molto simile a quello ad oggetti.

Le caratteristiche che distinguono questi sistemi gli uni dagli altri sono individuabili principalmente nel ruolo delle classi, nel meccanismo di comunicazione e di ereditarietà.

In SMALLTALK e LOOPS le classi sono degli elementi attivi che ricevono messaggi ed eseguono operazioni. La descrizione delle classi è quindi contenuta in metacassi (le quali a loro volta sono generate da una classe di sistema). Lo scambio di messaggi è di tipo sincrono. In SMALLTALK l'ereditarietà è un singolo genitore mentre in

LOOPS è multipla. Inoltre SMALLTALK non fornisce esplicitamente primitive per la combinazione dei metodi, primitive fornite invece in LOOPS.

In FLAVORS le classi sono elementi passivi, cioè con funzione puramente descrittiva (in modo analogo alla definizione di tipo di dato astratto). La comunicazione fra oggetti è sincrona, l'ereditarietà è multipla ed inoltre esiste un gran numero di primitive per la combinazione di metodi.

Infine nel modello ad attori le classi sono nuovamente di tipo passivo e la comunicazione è di tipo asincrono.
Come si vede esiste una varietà di linguaggi di programmazione con caratteristiche via via diverse da SMALLTALK fino al modello ad attori, passando attraverso LOOPS e FLAVORS.

SMALLTALK→LOOPS→FLAVORS→ATTORI

La scelta di implementare il formalismo dei Flavors è dovuta, da una parte alla volontà di accrescere l'ambiente Lisp disponibile su personal (GCLISP funzionante su Humming-Board, una scheda installata in un PC contenente un processore Intel 80386 e 6 megabyte di memoria) in modo da aumentare la sua compatibilità con l'ambiente lisp su Lisp-machine, nel nostro caso una TI-Explorer (il PC è stato poi effettivamente usato per implementare software destinato a "girare" sull'Explorer).

D'altra parte è stata anche determinante la caratteristica dei Flavors di essere una via intermedia fra Loops e gli attori e di rendere quindi possibili modifiche (non drammatiche) che consentano di inglobare anche le principali proprietà dei due sistemi "estremi". Ad esempio si può pensare di definire un flavor CLASSE e un flavor METACLASSE le cui istanze assumano un ruolo simile a quello di classi e metaclassi in Loops oppure di introdurre una funzione di scambio messaggi che consenta una comunicazione di tipo asincrono inviando i messaggi a un MONITOR che scheduli quindi i messaggi e li invii ai corrispondenti oggetti.

2. IDEE BASE PER L'IMPLEMENTAZIONE

Nell'implementazione di un Flavor system gli aspetti fondamentali riguardano la costruzione dI Flavors, intesi come descrizioni di classe, di istanze, di metodi e la costruzione di un meccanismo di comunicazione (sincrono) fra le istanze.

I Flavors sono componenti statiche che contengono informazioni riguardanti le instance-variables (il loro nome e l'eventuale valore di default) e i metodi a cui le istanze della classe rispondono. I dati contenuti consentono di risolvere il problema della ereditarietà di istanze e metodi.

L'ereditarietà è risolta in modo statico: il Flavor contiene le informazioni relative non solo alle instanze e i metodi dichiarati per la classe ma anche quele relative a tutte le variabili e i metodi che sono ereditati. Queste informazioni sono prese all'atto della creazione della classe e permettono di eliminare ricerche dinamiche negli alberi delle ereditarietà quando le istanze ricevono dei messaggi da eseguire. L'introduzione di nuovi metodi nelle classi viene notificata a tutte le classi superiori nell'albero di ereditarietà. La ridefinizione completa di una classe rende invece necessario che tutte le classi che ereditavano da questa siano ri-dichiarate.

Le istanze sono elementi attivi che ricevono dei messaggi. Il meccanismo di invio messaggi avviene tramite l'operazione di *send* eseguendo una applicazione funzionale dell'istanza sul messaggio e sui parametri. L'istanza è quindi una funzione che, mappata sui messaggi e sui parametri, produce un risultato.

ISTANZA(messaggio,parametri) → risultati

A differenza delle funzioni "normali" però l'istanza conserva un suo STATO (le instance variables) che non viene perso al termine della computazione, per cui in realtà l'applicazione della istanza diventa:

ISTANZA(stato,messaggio,parametri)→ risultati,nuovo-stato

Dove "messaggio" è mappato su un insieme di funzioni M (insieme dei metodi). L'istanza effettua il mapping per determinare la funzione corrispondente al messaggio (se c'è) e quindi applica questa funzione sullo stato e sui parametri.

M(messaggio)x(stato,parametri) →risultati,nuovo-stato

L'istanza diventa quindi l'associazione fra l'elemento attivo (la funzione di mapping) e uno passivo (lo stato):

ISTANZA = (FUNZIONE, STATO)

Per conservare lo stato del sistema è necessario che questo sia memorizzato in qualche luogo, inoltre le variabili dell'istanza devono essere legate ai loro valori alla chiamata dell'istanza e che al termine della computazione questi valori devono essere salvati. Questo meccanismo può essere realizzato sostanzialmente in due modi:

- l'istanza stessa al momento della sua chiamata effettua il binding delle variabili e al momento della sua uscita ne salva il valore;
- la funzione di invio del messaggio (SEND) si occupa di effettuare le operazioni di binding delle variabili e di salvataggio dei loro valori.

Entrambi i modi sono necessari a seconda dell'ambiente nel quale sono effettuati gli invii di messaggi. Infatti quando *send* è effettuata all'interno di una normale funzione Lisp è semplicemente necessario effettuare il binding delle variabili, mentre se *send* è effettuata all'interno dell'esecuzione di un metodo è anche necessario, prima di effettuare il binding delle variabili dello stato della istanza chiamata, anche salvare lo stato della istanza chiamante. Nel primo caso è sufficiente che sia l'istanza ad attuare lo stato e poi salvarlo, mentre nel secondo dovrà essere necessariamente *send* che dovrà salvare lo stato dell'istanza chiamante e quindi attuarlo nuovamente alla sua uscita. Questo è necessario sia perché in caso di chiamate ricorsive (self) il meccanismo dell'istanza per l'attuazione dello stato dovrà andare a recuperare dei dati aggiornati (appena salvati da *send*) sia per evitare problemi di ricoprimento delle instance variables quando si chiamino istanze della stessa classe o comunque con variabili con lo stesso nome.

Il meccanismo di attuazione e salvataggio degli stati diventa quindi:

ISTANZA	SEND
«attuazione dello stato»	«salvataggio dello stato dell'istanza
chiamate»	
.....	.....
-esecuzione della chiamata-	-chiamata all'istanza-
.....	.....
«salvataggio dello stato»	«attuazione dello stato dell 'istanza
chiamante»	

I *metodi* sono le funzioni che realizzano effettivamente la computazione. In un sistema a Flavor sono ereditati, in modo analogo all'ereditarietà delle istanze, dalle classi sottostanti nell'albero di ereditarietà. Inoltre esistono dei meccanismi di composizione dei *metodi* che permettono di specializzarli e di modificare il loro comportamento man mano che si sale nell'albero di ereditarietà. Ad esempio un metodo classico di composizione di *metodi* è quello":daemon" (di solito dato di default) che consente di stabilire delle operazioni da eseguire prima del body del *metodo* (:before) e dopo (:after). In questo modo un *metodo* ereditato può comportarsi diversamente, pur continuando ad effettuare le stesse operazioni di prima. Le modifiche effettuate saranno poi ereditate da questo punto in avanti e via via le varie classi potranno specializzare i *metodi* ereditati. L'ereditarietà è facilmente risolvibile con *metodi* simili a quelli delle instance-variables, memorizzando tutti i metodi a disposizione di un Flavor (ereditati e propri: quando viene creato un nuovo *metodo* la sua esistenza è segnalata ai Flavors superiori aggiornando i loro elenchi dei *metodi*) e mappando i *metodi* in modo da poter ricavare i nomi delle funzioni che effettivamente eseguono le operazioni. Più complicata è la risoluzione della composizione dei metodi. Un approccio standard è quello di copiare il body del *metodo*, combinarvi assieme le funzioni che ne modificano il comportamento e generare quindi una nuova funzione che esegua il tutto. Questo sistema ha lo svantaggio di portare a una duplicazione di codice ogni qualvolta si effettui una composizione di *metodi*. La soluzione scelta qui è diversa e uniforme con l'implementazione delle istanze. Un *metodo* è visto come una funzione A (attuatore) che applica (in un determinato ordine) un'insieme di funzioni F= {f}* sullo stato dell'istanza e sui parametri.

A(F,stato-stanza,parametri) → risultati,nuovo-stato

Come le istanze sono una coppia (funzione,stato) i *metodi* diventano così coppie:

(ATTUATORE,{f}*)

Dove ad esempio {f} è l'insieme delle funzioni di un metodo (before, body, after) e l'attuatore è una funzione che applica prima le funzioni :before, poi il body del *metodo* e quindi le funzioni :after. I vantaggi di questa rappresentazione sono:

- poiché {f} può essere un insieme di nomi di funzioni (e non di lambda-espressioni) l'ereditarietà e la composizione dei *metodi* si risolve in una composizione di insiemi evitando ridondanze e duplicazioni di codice;

- è facilmente risolvibile il problema di avere diverse modalità di method-composition (opzione :method-composition di defflavor): per ogni tipo di composizione è sufficiente definire un tipo di attuatore diverso ed assegnarlo ai *metodi* del Flavor;

- dato che le operazioni di composizione di *metodi* sono risolte in operazioni su insiemi di nomi di funzioni, ne consegue che le funzioni in sé possono facilmente essere compilate poiché non c'è bisogno di mantenere il loro body interpretato per poterlo poi manipolare nelle operazioni di composizione.

Per contro va notato che una rappresentazione del genere rende necessario un ciclo di valutazione in più e quindi un rallentamento del sistema. Inoltre l'efficienza del sistema

è avvertibile quando si creino *metodi* di dimensioni paragonabili a quelle dell'attuatore.

3. IMPLEMENTAZIONE

NOTA: L'implementazione proposta è totalmente sugli standard Common Lisp. In particolare non sono stati usati Stack-Group forniti su Zetalisp e GCLisp che comunque risultano particolarmente interessanti nell'ottica di fornire una comunicazione sincrona-asincrona in quanto permettono di salvare lo stato della computazione in un oggetto. E' quindi possibile pensare di trasportare il sistema anche su altri Common Lisp. Un'altra delle possibilità offerte dal GCLISP è quella della dynamic-closures, che risolvono automaticamente il problema del salvataggio dello stato delle istanze (le dynamic-closures sono binding di funzioni ed environments) ma sono al di fuori degli standard Common Lisp. Va d'altra parte notato che la versione GCLISP utilizzata era una versione ridotta di Common Lisp (mancando ad esempio della gestione di hash tables, utilizzate invece in ambiente Explorer per implementare i flavor) e quindi non si è potuto usufruire delle più avanzate "features" presenti nei "full Common Lisp".

DEFFLAVOR

Un Flavor è costituito da un simbolo con property-list contenente le seguenti informazioni:

: *instances-variables* : è la lista delle variabili locali al Flavor con i valori di default che assumono alla creazione delle istanze

((var1 val1) .. .. .. (varN valN)) :

:*methods-list* : è la lista di tutti i *metodi* appartenenti al Flavor (ereditati e no). Per distinguere i *metodi* ereditati da quelli posseduti localmente sono memorizzati i nomi dei Flavor proprietari dei *metodi*

((metodo1 flavor1 .. .. (metodoN flavorN))

I *metodi* sono funzioni il cui nome è costituito dal concatenamento del nome del Flavor e il nome del *metodo*. La :*method-list* serve anche per ricostruire questo nome.

:*lambda-form* : contiene il prototipo di lexical closure che costituirà l'instanziazione del Flavor. Contiene quindi una definizione di lambda-form generica (defun) in cui è lasciato indicato con **instance-name** il nome della funzione e con **instance-fun** il nome quotato (per problemi di puntatori) che è assegnato all'interno della funzione alla variabile SELF.

:*flavor-type* : ha sempre il valore Flavor e serve per testare che un simbolo sia un Flavor.

:*parent-of* : è l'elenco di tutti i Flavor che ereditano dal Flavor dato. (viene aggiornato al momento di creazione dei Flavor). Viene utilizzato per ereditare i *metodi*.

:*default-handler* : è utilizzata se il Flavor è creato con l'opzione ":default-handler". In tal caso contiene il nome della funzione da utilizzare quando non ci sono *metodi* per sviluppare l'operazione richiesta.

:*documentation* : contiene una stringa di documentazione.

Per risolvere il problema della creazione degli ambienti nei Flavors (creazione delle instance-variables e assegnazione dei loro valori) per ogni Flavor è creata una funzione di "swap" (parametri 'out e 'in) che si preoccupa di prendere i valori dal loro deposito "permanente" ("swap in") e assegnarli alle variabili e prendere i valori dalle variabili ("swap out") e ridepositarli. Queste operazioni sono fatte all'ingresso nel Flavor e all'uscita ed inoltre dalla funzione di *send* (per mantenere la consistenza degli ambienti anche in caso di chiamate ricorsive o comunque intrallaciate).

Il meccanismo è quindi:

ISTANZA
(<<funzione di swap>> 'in)
.....
....
SEND
(<<funzione di swap>> 'out)
<<esecuzione della chiamata>>
(<<funzione di swap>> 'in)
.....
....
(<<funzione di swap>> 'out)

La variabile globale **swap-variables**function contiene in ogni istante il nome della funzione di swap da utilizzare da *send* ed è aggiornata dinamicamente all'interno di ogni Flavor.

DEFMETHOD

I metodi sono costituiti da due componenti: la funzione che viene chiamata quando il Flavor riceve *send* del *metodo* e una struttura che contiene le operazioni vere e proprie che il *metodo* deve svolgere. La funzione svolge il ruolo di interprete di queste operazioni eseguendolein

sequenza: prima le operazioni" before" (quando sono stati creati uno o più *metodi* :before), poi "body" (la parte che viene creata con un defmethod normale) e quindi le operazioni "after" (quando sono stati creati uno o più metodi :after). Le operazioni sono contenute in tre property associate al nome della funzione :"before" (lista LIFO di tutte le operazioni di before), :body e :after (lista FIFO di tutte le operazioni after).

:before (lambda-before-1 lambda-before-2 ... ...)
:body lambda-body
:after (lambda-after-1 lambda-after-2)
property
METODO
functional-binding
(defun metodo ... ...
<< esegui operazioni before>>
<< esegui body >>
<< esegui operazioni after >>

MAKE-INSTANCE

Creazione delle istanze dei flavors e inizializzazione delle instance-variables. Viene restituito il simbolo a cui è associata la funzione di esecuzione dei *metodi*. Lo stesso simbolo ha in property-list le instance variables e il tipo di Flavor. La funzione che costituisce l'istanza è presa dal template di funzione contenuto nella property :lambda-form della definizione del Flavor.

:instance-variables ((var1 val1) ... ... (varn valn))
:flavor-type tipo-di-flavor
property
<FLAVOR_"tipo di flavor e numero di occorrenza">
functional-binding
(defun "tipo di flavor e numero di occorrenza" (message &rest parameters)
....
<<crea il bind fra le instance-variables e i loro valori memorizzati>>
<<esegui il metodo ricevuto come messaggio>>
<<memorizza i nuovi valori delle instance-variables>>

4. BIBLIOGRAFIA

[1] Briot, Cointe, THE OBJLISP MODEL: DEFINITION OF A UNIFORM, REFLEXIVE AND EXTENSIBLE OBJECT ORIENTED LANGUAGE, proceedings ECAI 1986

[2] Corradi, Leonardi, INTEGRAZIONE DEL PARADIGMA AD OGGETTI E AD ATTORI: PROPOSTA DI UN AMBIENTE DI PROGRAMMAZIONE NON TRADIZIONALE, Rivista di Informatica, Aprile/Giugno 1986

[3] Cox, OBJECT ORIENTED PROGRAMMING, Addison-Wesley 1986

[4] Leonardi, IL MODELLO AD OGGETTI: ANALISI DI ALCUNI AMBIENTI DI PROGRAMMAZIONE ESISTENTI, Proceedings AICA 1986

[5] Novak, GLISP: A LIST-BASED PROGRAMMING SYSTEM WITH DATA ABSTRACTION, The AI Magazine, Fall 1983

[6] Stefik, Bobrow, OBJECT ORIENTED PROGRAMMING: THEMES AND VARIATIONS, The AI Magazine, Winter 1985

[7] OBJECT-ORIENTED PROGRAMMING AND FLAVORS, Texas Instruments, Explorer System Lisp Reference

[8] USING FLAVORS, Texas Instruments, Explorer System Programming Primer

SOME RESULTS IN COMPLEXITY OF MATCHING, UNIFICATION, REWRITING PROBLEMS

Maria Paola Bonacina
Dipartimento di Scienze dell'Informazione
Università di Milano

RIASSUNTO

Questo lavoro è una rassegna sulla complessità computazionale di tre problemi di base nell'ambito della dimostrazione automatica di teoremi: matching, unificazione e riscrittura. E' strutturato in tre parti. La prima parte presenta le nozioni di base di teoria della complessità. Questa presentazione si limita a dare il significato intuitivo degli strumenti matematici applicati in teoria della complessità senza ripetere l'apparato formale dato in letteratura [3,14, 28]. Dopo aver brevemente rivisto i concetti di base relativi a questi problemi, i più recenti risultati su complessità di matching, unificazione e riscrittura sono riportati nella seconda parte. Infine nella terza parte un algoritmo di matching per la Rewrite Rule Machine [38] è descritto come un "pebbling game".

ABSTRACT

This work is an overview about computational complexity of three basic building blocks for automated theorem proving programs: matching, unification, rewriting. It is structured in three sections. The necessary background from complexity theory is presented in the first section. The aim of this presentation is not to rephrase the formal setting already given in the literature [3,14,8], but just to convey the intuitive meaning of the mathematical tools applied in complexity theory.
After briefly reviewing the basic concepts about these problems, the most recent results about complexity of matching, unification and rewriting are reported in the second section.
Finally in the third section a matching algorithm designed for the Rewrite Rule Machine project [38] is described as a pebbling game.

1. INTRODUCTION

Complexity theory is the study of computational complexity of computer algorithms. The computational complexity of an algorithm is measured by two fundamental factors: time and space. Roughly speaking, time measures how long it takes to perform the given algorithm onto a given input: it is the time elapsed between the beginning and the end of the algorithm execution. Space measures how much computer memory is required to perform the given algorithm onto a given input: it is the number of memory cells used during the algorithm execution.

Software development systems generally allow the programmer to know how much CPU time and how much central memory are used during any program run. An experiment about computational costs consists in running the program on some input and asking data about elapsed CPU time and allocated central memory to the system. Running a given program on a sample of inputs allows to collect some information about how running time and work space depend from some measure of the input, for instance its length.

The aim of complexity theory is to predict the results of these experiments. In order to be useful such a prediction should be done before that any effort to actually implement the algorithm is done: if the program running on a particular machine is already available there is no point in trying to predict its behaviour, it is much easier to experiment with it. The purpose of complexity theory is to predict time and space requirements of a given algorithm

before any program run, even before the abstract algorithm is translated in some language executable on real computers. In this way the theory can really help in choosing the right algorithm, i.e. the less expensive one. This will avoid the cost of implementing an algorithm that would be practically useless because it is much more time and pace consuming than desired.

This effort requires to establish an abstract model of computation defining precisely what is a machine, how an algorithm is given, what is the precise meaning of running time and work space. These definitions must be abstract in order to be independent of all the details introduced by the later choices of implementation language, software development system, working machine. In this way the abstraction level of complexity theory results is so high that they turn out to be a measure of how hard a given problem is.

2. PROBLEMS CLASSIFICATION IN COMPLEXITY THEORY

Problems can be classified in several ways. Complexity theory classifies problems depending on time and space requirements. However let's start first classifying problems as *decision problems*,*existence problems*, *counting problems* and *generation problems*. This classification will be useful in the following.
Next the basic notions of *upper bounds* and *lower bounds* and the most important *sequential and parallel complexity classes* will be introduced.

2.1 Problems Classification

Let A be the formal statement of some property. The basic types of questions about A we can ask are the following:

1. Decision problem:

given any x, does A(x) hold or not?

2. Existence problem:

does there exist an x such that A(x) holds?

3. Counting problem:

how many x's are there such that A(x) holds?

4. Generation problem:

enumerate all those x's such that A(x) holds.

This classification of course is not mandatory. An existence problem for instance can be rephrased as a decision problem: it amounts to decide the property $\exists x\ A(x)$.

A problem about a statement A can be solved in two ways: constructively and non constructively. Roughly speaking it is solved constructively if the individual x such that A(x) holds is actually constructed, while it is solved non constructively if an answer is given without exhibiting any x. This difference does not influence an algorithm to solve a decision problem: the decision algorithm takes as input the individual x and it answers "yes" if A(x) holds, "no" otherwise.

Choosing a constructive versus a non constructive approach does change instead the solution to an existence problem: in the constructive case the solution must exhibit the individual x such that A(x) holds, whereas in the non constructive case the answer will be either "yes, there exists such an x", without showing it, or "no, there is no such an x". In mathematics it is sometime useful to adopt the non constructive approach, because exhibiting an actual x is either too difficult or uninteresting. Computer algorithms are generally meant to be constructive, i.e. they answer constructing an element which satisfies the required property.

A counting problem is intrinsically non constructive, because it asks to know how many elements satisfy a given property without caring what they look like. However counting problems are usually given on combinatorial structures, where the individuals have a common regular structure, strings, monomials, and the single element does not bear any particular meaning.
Here it is much more interesting to get the cardinality of the set characterized by the given property, rather than pointing out the single elements, whose number can be very high.

Finally a generation problem can be considered the constructive version of a counting problem:solving a generation problem solves also a counting problem. Generation problems arise in contexts where the expected number of solutions is strongly lower than in most combinatorial problems. Moreover the actual structure of individuals is more meaningful than their number.

2.2 Upper bounds and lower bounds

Let's f and g be two functions on natural numbers. We say that f is an upper bound for g if

$$\exists r > 0 \exists n_0 > 0 \forall n \geq n_0\ g(n) \leq r f(n)$$

and we write this condition as $g = O(f)$.
This means that besides a finite number of exceptions, g is bounded to remain under f.

Let's suppose that g represents the running time to solve a problem A as a function of the input size f. Upper bounds are proved constructively: in order to establish that the running time for problem A is $O(f)$ it is necessary to exhibit an actual algorithm for A whose running time is f.
We say that f is an lower bound for g iff

$$\exists r > 0 \exists n_0 > 0 \forall n \geq n_0 \; g(n) \geq r f(n)$$

and we write this condition as $g = \Omega(f)$.
This means that besides a finite number of exceptions, g is bounded to remain over f.

Lower bounds are proved non constructively: in order to state that the running time for problem A is $\Omega(f)$, it is necessary to prove that no algorithm can solve A asking less than $f(n)$ running time on input sized n or equivalently any algorithm solving A will require a running time higher than f. No actual algorithm is needed. For this reason non trivial lower bounds are much more difficult to establish than upper bounds.

Let's suppose that an upper bound and a lower bound are given to A's running time. Working on the upper bound means trying to design a faster algorithm in order to lower the upper bound. Working on the lower bound means trying to get a better understanding of problem A in order to raise its lower bound. The optimal running time for A lies somewhere between the lower bound and the upper bound: getting them closer to each other means having a better approximation to the optimal solution.

The same considerations apply to working space. It can happen that an algorithm allows to lower a time (space) upper bound consuming more space (time) than the previously known algorithm. Great care must be taken to build a good compromise between time and space.
Such a trade-off can usually be determined only taking also into account the particular nature of the problem and the algorithm applications.

2.3 Sequential Complexity Classes

In order to have this report as much self-contained as possible let's recall the most well known complexity classes.

Complexity classes are introduced to give a problem classification based on resources requirements. Resources are *running time* and *work space*.
Running time and work space of a Turing machine M are partial functions from N to N. They are partial functions, because the machine can miss to halt on some input and in this case its running time and work space are undefined.

The running time t_M of a deterministic Turing machine M is the partial function from N to N such that $t_M(n)$ is the maximum computation time of M on an input of length n, if M halts on all of them, otherwise it is undefined [3]. The computation time is simply the number of machine steps. Analogous definitions are given for space and for the nondeterministic machine [3].
The key point is choosing the maximum computation time among all computation times on n sized inputs. Saying that t_M is the running time of M does not imply that given any input x M will eventually halt on c in $t_M(|x|)$ steps. It implies M will eventually halt on x in at most $t_M(|x|)$ steps. Taking the maximum means considering the worst case: complexity classes give a worst case classification. This point has to be taken into account when complexity theory results are applied to forecast or to evaluate practical results.

Complexity classes are finally defined as follows [3]:
- $A \in$ DTIME (t) iff A is recognized by a deterministic Turing machine whose running time is $O(t)$.
- $A \in$ NTIME (t) iff A is recognized by a non deterministic Turing machine whose running time is $O(t)$.
- $A \in$ DSPACE (t) iff A is recognized by a deterministic Turing machine whose working space is $O(t)$.
- $A \in$ NSPACE (t) iff A is recognized by a non deterministic Turing machine whose working space is $O(t)$.

The general concept of "solving a problem" is rephrased unambiguously as "recognizing a formal language". Language recognition is a decision problem: decision problem is assumed as the basic problem model in complexity theory.
The central complexity classes are:

$$LOG = \bigcup_{c \geq 1} DSPACE(c * logn)$$

$$NLOG = \bigcup_{c \geq 1} NSPACE(c * logn)$$

$$P = \bigcup_{i \geq 0} DTIME(n^i)$$

$$NP = \bigcup_{i \geq 0} NTIME(n^i)$$

$$PSPACE = \bigcup_{i \geq 0} DSPACE(n^i)$$

$$NPSPACE = \bigcup_{i \geq 0} NSPACE(n^i)$$

$$DEXT = \bigcup_{c \geq 0} DTIME(2^{c*n})$$

$$NEXT = \bigcup_{c \geq 0} NTIME(2^{c*n})$$

These classes are called sequential because the underlying abstract machine, the Turing machine, works sequentially. A deterministic Turing machine specifies step by step a single computation history. A nondeterministic Turing machine specifies several computation histories: it can be realized either by a sequential machine performing one computation thread at a time and backtracking when it fails to reach the goal, or by a parallel machine developing in parallel all the computation threads.
Due to the different underlying machine deterministic and nondeterministic classes have a different intuitive meaning. Let's consider the polynomial case: if $A \in P$, then A can be solved in polynomial time by an algorithm on a real computer, since a deterministic Turing machine is just a very high abstraction of a real machine. Of course this does not mean that A will be solved easily, because the leading n's power in the polynomial can still be very high and moreover the coefficient # in the definition of upper bound can be very high as well. However the problem is tractable and these factors can eventually be reduced by more clever algorithm design.
The meaning of $A \in NP$ is much less intuitive because of nondeterminism. If $A \in NP$ then it is not known how to compute a solution to A in polynomial time, but it is known how to check in polynomial time whether a given solution is indeed a solution to A.

2.4 Problem Reducibility and Completeness

The notion of reducibility allows to compare problems with respect to their difficulty.
Given two problems A_1 and A_2, A_1 is *polynomial time many-one reducible* [3] to A_2 iff there exists a function f computable in polynomial time such that

$$\forall x \; x \in A_1 \text{ iff } f(x) \in A_2$$

This is written shortly as $A_1 \leq_m A_2$
The intuitive meaning is that A_1 is someway easier than A_2. Namely if A_2 is solvable, then A_1 will be solvable as well just transforming the problem $x \in A_1$ into the equivalent problem $f(x) \in A_2$. The transformation will cost polynomial time so that if A_2 is in P, A_1 will be in P as well.

Having this ordering between problems it is natural to ask which are the most difficult problems inside a complexity class C.
Given a class C, a problem A_1 is *C-hard* iff $\forall A_2 \in C$, $A_2 \leq_m A_1$ holds: a problem A_1 is *C-complete* (for polynomial time reducibility) iff it is C-hard and $A_1 \in C$.
C-complete problems are the most difficult problems in class C, since any other problem in C is reducible to them. It follows that studying these problems is expected to give a better understanding of the whole class.

The above definition of reducibility can be modified putting different requirements on the function f: for instance if f is required to be computable in logarithmic space, we obtain the definition of *logspace many-one reducibility* and the related notion of *logspace completeness*.

2.5 Parallel Complexity Classes

In order to formalize complexity of parallel algorithms, parallel complexity classes are defined. The underlying abstract machine is called PRAM, standing for Parallel Random Access Machine. It is given by a pool of processors working in parallel and sharing a common random access memory. Since this machine is very unrealistic many restrictions to memory access have been designed to yield several similar abstract machine:

- CRCW: Concurrent Read Concurrent Write
- CREW: Concurrent Read Exclusive Write
- ERCW: Exclusive Read Concurrent Write
- EREW: Exclusive Read Exclusive Write

However the basic parallel complexity class is based on the unrestricted PRAM, i.e. the CRCW PRAM: $A \in NC$ [41] iff A is solved by a PRAM with $P(n) = n^{O(1)}$ and running time $\log^{O(1)} n$, where $P(n)$ is the number of processors and n is the input size.

Intuitively a problem in NC is a tractable problem, indeed $NC \subseteq P$, which takes advantage of parallel execution. The sequential polynomial running time becomes a polylogar-

ithmic parallel running time if $n^{O(1)}$ are provided.

The notion of *speed-up* [1] is introduced to express the advantage given by parallel execution:

$$speed\text{-}up\ (n) = \frac{\text{TIME}_{PRAM}\ (n)}{\text{TIME}_{RAM}\ (n)}$$

where RAM is just the sequential abstract machine corresponding to the PRAM model.

The speed-up is the ratio between the parallel algorithm running time and the faster known sequential algorithm running time. A problem belonging to *NC* is a problem whose speed-up is $O(\frac{log\ (1)\ n}{n^{i}})$. The *NC* class has been criticized [58] because it asks for a dramatic gain in running time at the expenses of a quite unrealistic number of processors. There are problems in *P*, which could exploit parallelism, even if they do not belong to *NC*: $A \in PC$ [58] iff *A* is solved by a PRAM with $P(n) \in O(n^{O(1)})$ and such that for any RAM solving A the following holds:

$$\lim_{n \to +\infty} \frac{\text{TIME}_{PRAM}\ (n)}{\text{TIME}_{RAM}\ (n)} = 0$$

$A \in PC^{*}$ [58] iff A is solved by a PRAM with $P(n) \in O(n^{O(1)})$ and such that for any RAM solving A the following holds:

$$\lim_{n \to +\infty} \frac{P(n)\ \text{TIME}_{PRAM}\ (n)}{\text{TIME}_{RAM}\ (n)} = O(1)$$

A problem belongs to *PC* if its sequential running time is speed-up by more than a constant factor using a polynomial number of processors.
A problem belongs to *PC** if the speed-up is proportional to the number of processors.
All these definitions involve both running time and processors number: when switching from sequential to parallel algorithms, the trade-off between running time and working space is substituted by the trade-off between running time and number of processors.

3. MATCHING, REWRITING, UNIFICATION

This section reports some results obtained applying the tools of complexity theory to the matching, rewriting, unification problems.
These problems are chosen because they are the basic building blocks of all the most widely used inference rules in automated reasoning. Programs for automated reasoning are generally very expensive in both time and space, so that it is particularly important to know a priori the degree of complexity for at least these basic building blocks.

Moreover the increasing interest toward parallelism has lead to consider to what extent these basic steps can be parallelized: again complexity theory is a good tool to discover something about the sequential or parallel nature of a problem.

3.1 Problems definition

Let's start with some basic definitions involved in the problems at hand.
In the following X, F, $T(X,F)$, $T(F)$ are respectively a set of variable symbols, a set of function and constant symbols, the universe of finite terms and the universe of ground finite terms generated from X and F. Let $C \subseteq F$ identify the set of constant symbols.
A term where every variable appears at most once is called *linear*.
An assignment is a function $\sigma : X \rightarrow T(X,F)$ such that its *domain* $(\sigma) = \{x\ \sigma(x) \neq x\}$ is finite.
An assignment σ extends to a *substitution* $\sigma' : T(X,F) \rightarrow T(X,F)$ as follows:

1. $\sigma'(x) = \sigma(x)$ if $x \in X$

2. $\sigma'(c) = c$ if $c \in C$

3. $\sigma'(f(t_1,\ldots,t_n)) = f(\sigma'(t_1),\ldots,\sigma'(t_n))$ if $f \in F$ with arity $n \geq 1$ and $t_i \in T(X,F)$, $1 \leq i \leq n$.

Many authors adopt postfix notation for substitution. Here and in the following we maintain the classical functional prefix notation.

Substitution composition works as follows: let $\vartheta = \{x_1 \leftarrow t_1,\ldots, x_n \leftarrow t_n\}$ and $\lambda = \{y_1 \leftarrow u_1,\ldots,y_m \leftarrow u_m\}$ be two substitutions.

The composition $\lambda \circ \vartheta$ is obtained from the set

$$\{x_1 \leftarrow \lambda(t_1),\ldots, x_n \leftarrow \lambda(t_n), y_1 \leftarrow u_1,\ldots,y_m \leftarrow u_m\}$$

by deleting any element $x_i \leftarrow \lambda(t_i)$ for which $x_i = \lambda(t_i)$ and any element $y_j \leftarrow u_j$ such that y_j is among $\{x_1,\ldots, x_n\}$ [10].

A substitution σ is *idempotent* iff $\sigma^{\circ}\sigma = \sigma$.
It can be proved [32] that a substitution $\sigma = \{ x_i \leftarrow t_i \}$ $1 \leq i \leq n$ is idempotent iff $D(\sigma) \cap R(\sigma) = 0$ where $R(\sigma) = \{ y | y \in X, y \text{ occours in } t_i \text{ for some } i, 1 \leq i \leq n \}$ is the *range* of σ. From now on we restrict our attention to idempotent substitutions. This assumption is quite natural in theorem proving, where substitutions involve terms belonging to different formulas, for instance different clauses or different rewrite rules. Assuming every formula or term has distinct variables implies substitutions are idempotent because their domains and ranges are disjoint.

An equation is a pair of terms $l \leftrightarrow r$; a *rewrite rule* is an oriented equation $l \leftrightarrow r$, such that $Var(r) \subseteq Var(l)$, where $Var(r)$ is the set of variables occurring in the term r.
An *equational theory* E is a set of equations $\{l_i \leftrightarrow r\}_{i=1}$.
The relation $\leftrightarrow$ is transitive, symmetric and reflexive.
Let $\underset{E}{\leftrightarrow}$ denote the smallest symmetric, stable and monotonic relation containings E and $\overset{\bullet}{\underset{E}{\leftrightarrow}}$ the reflexive transitive closure of $\underset{E}{\leftrightarrow}$

The matching and rewriting problems

Given an equational theory E and two terms s and t, does there exist a substitution σ *such that* $\sigma(s) \overset{\bullet}{\underset{E}{\leftrightarrow}} t$? Often this problem is called *E-matching*, reserving the name *matching* for the simplest case where E is empty. The term s is called *pattern* and the term t is called *subject*.
If the pattern is a linear term the matching problem is called *Linear matching*.
Given a term s and a rewrite rule $l \rightarrow r$, the *rewriting problem* consists in

- deciding if there are a subterm $s\ u$ and a matching substitution σ such that $s\ u = \sigma(l)$;

- replacing s/u by $\sigma(r)$ getting $s' = s\ [u \leftarrow \sigma(r)]$.

The rewriting problem is solved solving a matching problem and then performing a replacement step.

The unification problem

Given an equational theory E and two terms s and t, does there exist a substitution σ *such that* $\sigma(s) \overset{\bullet}{\underset{E}{\leftrightarrow}} t$?

Against this problem is often called *E-unification*, reserving the name *unification* for the simplest case where E is empty.

If the two terms to be unified are both linear the unification problem is called *linear unification*.

For the purposes of automated theorem proving this problem is refined to ask that the unifier σ is the most general.
An E-unifier τ for s and t is an *E-most general unifier* (E-mgu) iff
$\forall \sigma$ *such that* $\sigma(s) \overset{\bullet}{\underset{E}{\leftrightarrow}} \sigma(t)\ \exists \sigma_0$ *such that* $\sigma = \sigma_0 \circ \tau$ [10].

A unifier is a most general unifier for s and t iff it is a maximal general element in the set of all unifiers of s and t. Being maximal is defined with respect to the following ordering relation between substitutions:

$$\tau > \sigma \text{ iff } \exists \sigma_0\ \sigma = \sigma_0 \circ \tau$$

This intuitively means that τ is more general than σ, since σ can be obtained from τ by further instantiating its range. When E is empty the mgu for two terms is unique up to variable renaming, but for a general E it is not unique, so that the unification problem must refer to the set of all maximal general unifiers instead than to the most general one [18].

Let's introduce some naming conventions, which are very useful to talk about unification algorithms. Unifying $s = f(s_1,....,s_n)$ and $t = g(t_1,....,t_n)$ requires that f and g are the same function symbol and that the corresponding subterms unify, i.e. $\forall 1 \leq i \leq n$ s_i and t_i unify. Two terms can fail to unify in two ways. The first one is called *clash failure*: two terms fail to unify because they have different function or constant symbols at corresponding positions. The second one is called *occur failure*: two terms fail to unify because a variable x appearing in one of them gets instantiated to a term where x itself occurs. The obtained substitution is no more idempotent, because x belongs to both its domain and its range. This circularity would require to deal with infinite subterms. Unification algorithms generally involve a so called *occur check* phase where the candidate unifier is checked to discover these failures.

Both the matching and the unification problems are existence problems. They are intended to be solved constructively: a matching and a unification algorithms are expected to yield the required substitutions. The rewriting problem can be viewed as a generation problem, since the algorithm performing rewriting is able to generate all those terms to which the given term s can be rewritten to.

3.2 The matching problem

The following notations are used to identify matching and unification problems:

A: associativity, $f(f(x,y),z) = f(x,f(y,z))$
C: commutativity $f(x,y) = f(y,x)$
DL: left distributivity, $f(x,g(y,z)) = g(f(x,y), f(x,z))$
DR: right distributivity, $f(g(x,y),z) = g(f(x,z) f(y,z))$
(All results about DL carry over to DR)
D: left and right distributiviry
I: idempotence, $f(x,x) = x$
U: unit element, $f(x,u) = f(u,x) = x$

For instance the A+C-matching (unification) problem is the E-matching (unification) problem in the theory E containing the associativity and commutativity axioms for function symbols in the signature.

Results about matching problems are summarized in the following tables. First complexity classes for matching problems on equational theories and finitely presented algebras are listed.

Next the actual sequential and parallel upper bounds for both linear matching and matching are given. Two recent parallel lower bounds follow.

Complexity Classes for Matching Problems:

Problem	*Complexity Class*	*Reference*
Matching	*NC*	*[11]*
U-matching	*NP-complete*	*[2]*
D-matching	*NP-hard*	*[2]*
D+U-matching	*NP-hard*	*[2]*

where the results are given in chronological order. Set matching and bag matching and similarly set unification and bag unification are the matching and unification problems applied to sets and bags of terms rather than single terms. Set matching and bag matching are considered the simplest cases of A+C+I-matching and A+C-matching respectively. The same holds for the corresponding unification problems.

Almost all of these results are NP-completeness results.

Problem	*Complexity Class*	*Reference*
A+C-matching	*NP-complete*	*[5,6]*
Linear A+C-matching	*P*	*[5,6]*
C+I-matching	*NP-hard*	*[20]*
A+I-matching	*NP-hard*	*[20]*
Set matching	*NP-complete*	*[20]*
A+C+I-matching	*NP-complete*	*[21]*
A+C+U-matching	*NP-complete*	*[21]*
A-matching	*NP-complete*	*[6]*
C-matching	*NP-complete*	*[6]*
I-matching	*NP-complete*	*[22]*
C+U-matching	*NP-complete*	*[22]*
A+U-matching	*NP-complete*	*[22]*
Bag-matching	*NP-complete*	*[22]*

They are generally obtained first showing that the given matching problem is NP-hard and then showing that it is reducible to a problem which is already known to be NP-complete.

The basic matching problem is tractable and moreover it is in the NC class, i.e. it is efficiently parallelizable. If properties are added to the function symbols, the problem turns out to be untractable. Moreover it does not seemthatvarying the assumed properties causes any change in complexity. The only exception is given by linear A+C-matching. It belongs to the P class, i.e. it is tractable, whereas the unrestricted A+C-matching problem is NP-complete. As the following treatment will confirm, the linearity assumption generally reduces the complexity of both unification and matching problems. If the pattern is linear, there is no need to ensure that all the occurrences of the same pattern variable get instantiated to the same term, since every variable occurs once.

Complexity Classes for Matching Problems in Finitely Presented Algebras:

Problem	*Complexity Class*	*Reference*
FPCSG	*decidable*	*[22]*
FCSG	*NP-complete*	*[21]*
FCSGI	*P*	*[22]*
FCMI	*P*	*[22]*
FPAG	*P*	*[7,30,31,19]*
FBR	*NP-complete*	*[8,37,19]*
FPBR	*NP-hard*	*[19]*
FPA	*NP-complete*	*[27]*
SR	*NP-complete*	*[22]*

where [22]:

FPCSG: Finitely Presented Commutative Semigroup

FCSG: Finitely Generated Free Commutative Semigroup

FCSGI: Finitely Generated Free Commutative Semigroup with Idempotency

FCMI: Finitely Generated Free Commutative Monoids with Idempotency

FPAG: Finitely Presented Abelian Group

FBR: Finitely Generated Free Boolean Ring

FPBR: Finitely Presented Boolean Ring

FPA: Finitely Presented Algebra

SR: theory presented by a finite canonical term rewriting system, such that for each rule the right had side is either a ground term or a subterm of the left hand side.

A finitely presented algebra is presented by a finite set of generators, a finite set of relations expressed using generators and the operator symbols of the algebra. A freely generated algebra is a finitely presented algebra without relations.

Sequential Algorithms for Matching:

Problem	*Time*	*Reference*
Linear matching	$O(2^{\|s\|} + \|t\|)$ $O(\|s\| * \|t\|)$ $O(2^{\|s\|})$	*[16]* *[16]* *[8]*
Matching	$O(\|s\| + \|t\|)$	*[40]*
Linear AC - matching	$O(\|s\| * \|t\|^3)$	*[5,6]*

where

- $|s|$ is the size of the pattern s
- $|t|$ is the size of the subject t
- the linear bound given in [40] works only if *depth* (s) = *depth* (t) = 1, i.e. both terms are completely flattened
- the $O(2^{|s|} + |t|)$ upper bound is obtained by a bottom-up algorithm, which has been refined in [7] to yield the upper bound $O(2^{|s|})$
- the $O(|s| * |t|)$ upper bound which lets the problem belong to P is obtained by a top-down algorithm.

Algorithms for linear matching are very useful. If no function symbols properties are given, a linear matching algorithm can be extended to a matching algorithm without modifying the upper bound. This has been done for instance in [43] extending to arbitrary matching the last result in our table. Moreover a matching algorithm often contains a linear matching algorithm as a preprocessing phase. If the pattern is indeed linear the preprocessing phase will suffice. If the pattern is not linear the linear algorithm allows to discover as soon as possible some non matching situations.

Parallel Algorithms for Linear Matchings:

Machine	*Time*	*No. of Processor*	*Imput size*	*Reference*
CREW PRAM	$O(\log^2 n)$	$O(n^5)$	$n = \|s\| + \|t\|$ s and t as dags	*[11]*
PRAM	$O(m_2)$	$O(m_1)$	$m_1 = \|s\|$ $m_2 = \|t\|$	*[38]*
EREW PRAM	$O(n^{\varepsilon}. \log n)$	$O(n^{2-\varepsilon})$	$n = \|t\|$ *as tree* $\varepsilon \geq 0$	*[45]*
CREW PRAM	$O(\log^2 n)$	$O\left(\frac{n.k}{\log^2 n}\right)$	$n = \|t\|$ *as tree* $k = \|\text{Var}(s)\|$	*[45]*
CREW PRAM	$O(\log n)$	$O\left(\frac{n.k}{\log n}\right)$	$n = \|t\|$ *as tree* $k = \|\text{Var}(s)\|$	*[45]*

where again:

- $|s|$ is the size of the pattern s

- $|t|$ is the size ofthe subject t

These results are unfortunately quite difficult to compare since they even exploit different parallel machine models and different term representation structures (trees or dags). The first algorithm listed here [11] had an impor tant impact on research, since it was one of the early results showing that matching is indeed parallelizable. However its $O(n^5)$ requirement on the number of processors has been considered unrealistic since the algorithm appeared. The algorithm given in [45] on the same machine model lowers the number of processors from $O(n^5)$ to $O(\frac{n*k}{\log 2n})$. If concurrent writes are allowed, the time upper bound is lowered from $O(\log^2 n)$ to $O(\log n)$ with the same number of processors.

Unfortunately all these PRAM models are quite unrealistic. The only result not actually relying on PRAMs is the one given in [38]. Here its bounds are given for the general PRAM in order to allow some rough comparison with

the other results. However this algorithm is designed for the RRM (Rewrite Rule Machine) a highly parallel special purpose machine to execute rewriting based languages. The third section is devoted to explain in details this algorithm.

Parallel Algorithms for Matching:

Machine	*Time*	*NO. of Processor*	*Imput size*	*Reference*
CREW PRAM	$O(\log^2 n)$	$M(n)$	$n = \|s\| + \|t\|$ *as dags* $M(n) \approx n^{2.49}$	*[13]*
CREW PRAM	$O(\log n)$	$O(n^3)$	$n = \|s\| + \|t\|$ *as trees*	*[57]*

where $M(n) \approx n^{2.49}$ is the best known sequential bound for multiplying two $n \chi n$ matrices.

These last two results need some comments. The first one [13] is still the best one as far as processors number is concerned. This algorithm is a parallel randomized algorithm. The definition of randomized algorithm is given in [44]: a sequential randomized algorithm is an algorithm such that the processor can flip at each step an unbiased coin and take a decision based on the flip outcome. In the parallel case [25] all processors can flip coins independently. Randomization modifies the usual notion of algorithm. Let's consider the problem to decide the property A. An algorithm to decide A answers "yes", if $A(\chi)$ holds, "no", if $A(\chi)$ does not hold.

A randomized algorithm as in [12] decides A as follows: it answers "yes", if $A(\chi)$ holds; if $A(\chi)$ does not hold, it answers "yes" with probability $p_E \leq \varepsilon$ and "no" with probability $1 - p_E$, where $\in$ is a constant such that $0 \leq \varepsilon \leq 1$. Moreover if the execution is repeated c times p_E decreases to at most 2^{-c}.

The second result [57] is the most recent one and it is very important because in [57] it is shown that it is optimal, i.e. $O(\log n)$ is also the lower bound for matching on the CREW PRAM:

3.3 The rewriting problem

A rewriting step is given by a matching step and a replacement step, so that the rewriting complexity depends from the matching cost. The replacement cost, i.e. the cost of term reshaping strongly depends from term representation.
Term representation choices generally obeys either the *structure copying* approach or the *structure sharing* approach. Roughly speaking, in the first case different occurrences of the same substructure are repeated, whereas in the second one they are shared. Terms are represented as trees in the structure copying approach, whereas structure sharing is obtained using dags, i.e. letting common subterms be shared by more than one parent node instead than repeating them [12]. More precisely a dag is

Lower bounds for Matching:

Machine	*Time*	*No. of Processor*	*Impuz size*	*Reference*
CRCW PRAM	$\Omega\left(\frac{\log n}{\log \log n}\right)$	$O(n^i)$	$n = \|s\| + \|t\|$ *as dags or tress* $i \geq O$	*[57]*
EZREWor CREW PRAM	$\Omega(\log n)$	$O(n^i)$	$n = \|s\| + \|t\|$ *as dags or tress* $i \geq O$	*[57]*

a direct acyclic graph; a tree is a dag such that each node has at most one incoming arc, i.e. at most one parent node. Structure copying consumes more space, but it gives a faster access to terms. Structure sharing saves memory at the expense of a more complex access: for instance, since a subdag is referred by possibly many parents, a reference counting method is needed. How much memory is saved through structure sharing depends from the term structure: in the best case it will be exponential, but this is quite rare [15].

The most well known issues about rewriting on trees versus rewriting on dags are related to non linearity [4,12,15]:

- "test for equality of subterms": the need for this test arises when the pattern is a non linear term, for instancex + x from the rewrite rule x+x→x. Linear matching is intrinsically simpler than matching, because there is no need to test that different occurrences of the same pattern variable get the same subterm. This test is much easier on dags, where repeated subterms are represented as the same subdag.
- "number of copies": when a non right linear rule such as $x*(y+z) \rightarrow (x+y)+(x*z)$ is applied, a variable, here x, gets duplicated. Structure copying requires duplicating the whole subterm which x is istantiated to, while structure sharing just requires to set a pointer.

3.4 The unification problem

Problem	*Decidable*	*Reference*
A-unification	*YES*	*[42,34,50]*
D+A-unification	*NO*	*[56]*
D+C-unification	*?*	*[56]*
D+A+C-unification	*NO*	*[56]*
D+A+I-unification	*YES*	*[56]*
DL-unification	*YES*	*[2]*
A+DL+U-unification	*NO*	*[2]*
A+U-unification	*YES*	*[22]*

Results about unification are organized as follows. The first table summarizes those unification problems, for which either it is known that they are undecidable or it is known that they are decidable, but the complexity is not known. Unification problems already assigned to a complexity classes are listed in the second and third tables: the first is devoted to problems on equational theories and the second one collects results about finitely presented algebras. The fourth table shows known sequential upper bounds for the unification problem. Finally we report some results about the debated question of parallel unification.

It seems that the distributivity axiom lets significantly more difficult to prove something more than decidability. Moreover it has to be noticed that whereas A-matching is NP-complete, A-unification is only known to be decidable and primitive recursive.

The second table is about those unification problems known to belog to a complexity class:

Complexity Classes for Unification Problems:

Problem	*Complexity Class*	*Reference*
Unification	*logspace complete in P*	*[11,60]*
Linear unification	*logspace complete in P*	*[13]*
C-unification	*NP-complete*	*[14]*
D-unifieation	*NP-hard*	*[2]*
U-unification	*NP-complete*	*[2]*
D+U-unification	*NP-hard*	*[2]*
C+I-unification	*NP-hard*	*[20]*
A+I-unification	*NP-hard*	*[20]*
Set unification	*NP-complete*	*(20)*

Problem	*Complexity Class*	*Reference*
A+C-unification	*NP-complete*	*[21]*
A+C+U-unification	*NP-complete*	*[21]*
A+C+I-unification	*NP-complete*	*[21]*
Linear A+C-unification	*P*	*[21]*
I-unification	*NP-complete*	*[22]*
C+U-unification	*NP-complete*	*[22]*
Bag unification	*NP-complete*	*[22]*

where the results are given in chronological order. The U-unification problem is given over a signature including other constants besides the unit element [2].
These results confirm the previous table on complexity classes of matching problems. Adding axioms let the problem become untractable Varying the set of axioms does not seem to have remarkable effects. Linearity helps significantly. Unification and linear unification belong to the same complexity class, even if linear unification is practically much more easier. If the terms are linear, there is no need to ensure that all the occurrences of the same variable get instantiated to the same term, since every variable occurs once. Moreover no occur check is required, because it cannot happen that a variable x in s gets instantiated to a subterm from t containing a variable y already instantiated to a subterm from s where there is another occurrence of x. The A+C problem emphasizes this difference: the linear case is in P, whereas the unrestricted case is NP-complete.

Complexity Classes for Unification Problems in Finitely Presented Algebras:

Problem	*Complexity Class*	*Reference*
FPCSG	*decidable*	*[22]*
FCSG	*NP-complete*	*[21]*
FCSGI	*P*	*[22]*
FCMI	*P*	*[2]*
FPAG	*P*	*[7,30,31,19]*
FBR	*NP-complete*	*[8,37,19]*
FPBR	*NP-hard*	*[19]*
FPA	*NP-complete*	*[27]*
SR	*NP-complete*	*(22)*

See the corresponding table for matching problems for the notations.

Sequential Algorithms for Unification:

Time	*Input*	*Reference*
$O(2^n)$	$n = \|s\| + \|t\|$	*[47]*
$O(n + m.logm)$	$n = \|s\| + \|t\|$ $m = \|Var(s)\| + \|Var(t)\|$	*[37]*
$O(n)$	$n = \|s\| + \|t\|$ $depth(s) = depth(t) = 1$	*[40]*

The result given in [40] requires that the two terms have both depth 1, i.e. they are both completely flattened.

While matching belongs to the parallel complexity class NC, unification is considered inherently sequential since it is logspace complete in P as shown independently in [11] and in [60].
The first result has been obtained working on trees, while the second one has been obtained working on dags. This result has been extended to linear unification on trees in [13]. Since unification is complete for P, if it were in NC then $NC = P$, which popularly unlikely.

However it has been shown in [58] that unification on non sparse graphs is in PC and also in PC^*, while it is still an open problem whether unification is in PC. A non sparse graph is a graph such that $E = \Omega(f(V))$, where E is the number of edges, V is the number of nodes and f is a monotone increasing function $f : N \rightarrow N$ such that $\lim_{n\to+\infty} \frac{f(n)}{n} = 0$. Intuitively a graph is sparse if it is poorly

Problem	*Machine*	*Time*	*No. of Processor*	*Reference*
Unification	*CRCW PRAM*	$O(\frac{E \log^* E + V}{P})$	$O(E \log^* E)$	*[58]*
Linear unication	*EREW PRAM*	$O(\frac{E}{P} + V \log P)$	$O(E \log^* E)$	*[58]*

4. A MATCHING ALGORITHM

This third session presents an algorithm for parallel matching using "pebbling" as informal algorithm presentation technique.
An algorithm for parallel matching is given in [38] as one of the first algorithms especially designed for the Rewrite Rule Machine (RRM) project [15,59].
The complexity of this algorithm on the PRAM has been listed in the above table about parallel algorithms for linear matching. However this algorithm has been designed in order to perform very efficiently on the RRM, so that comparisons made referring to the PRAM could be misleading. The basic architectural features mirrored by the algorithm are *active memory* and *local communication*. As we will see the algorithm is designed in order to exploit very tiny processing elements performing in parallel very simple instructions and mainly communicating to their closest neighbours. This algorithm is specified as an abstract SIMD (Single Instruction Multiple Data) machine, called MM. Each node in a term is conceived as an abstract PE (Processing Element). At each step the same instruction is performed on all the nodes.

MM performs parallel matching of a set of patterns with respect to a single subject term: it is the so called *many to one* matching. Many to one matching arise when matching is required in order to rewrite: the term to be rewritten is generally one, whereas the rewrite rules to reduce it are many. Every left hand side of a rewrite rule yield a pattern. In the following the subject term will be called term when the meaning is clear from the context. The MM described in [38] handle only linear patterns. The RRM will need unrestricted matching. However implementing linear matching before than matching is useful, because a matching algorithm often contains a linear matching algorithm as a preprocessing phase. If the patterns are indeed linear the preprocessing phase will suffice. If the patterns are not linear the linear algorithm will apply to discover as soon as possible some non matching situations.

Let's represent terms as trees. In the real RRM they will probably represented as dags, but trees are easy for an intuitive presentation. The subject term represent the input data to the MM. It is spread over the PEs is such a way that each PE holds a term node. A single PE is a very tiny processor: its memory holds just a term node and some *pebbles* and its instruction set provide only few very simple operations to be performed onto pebbles. They resemble much more to active memory cells than to actual processors.

The patterns represent the program given to the MM: rewrite rules are compiled in the RRM code and then the program part encoding left hand sides is given to the MM.

During the execution pebbles at a node represent which patterns or just (sub)patterns match that node. Patterns are matched to the term working bottom up, so that at an intermediate stage it can be known that a subpattern matches, but it is not yet known if the whole pattern matches.
When the computation halts pebbles at a node represent which patterns match that node, so that the final pebbles state encodes the matching substitution.

The algorithm is described here informally as a *pebble game*. Pebbles games as an algorithm presentation tool will be adressed by the next paragraph.
The game is played on two sides, the term side and the patterns side. The state of the game is given by the distribution of the pebbles. A single pebble p_i can occur at only one node on the pattern side, whereas it can occur at more nodes on the term side. That means several subterms in the term correspond to the same (sub)pattern or equivalently this same (sub)pattern matches several subterms in the subject term. When ever a pebble is moved on the patterns side all its occurrences on the term side are moved accordingly: the patterns really drive the search for a match and the pebbles distribution on the term mirrors the state of the search.

The following game rules have been abstracted from examples given in [38]:

Pattern side (working in parallel on each pattern):

If the current pattern node is a leaf, then
if it is labelled by a variable symbol, then go to its parent
if it is labelled by a constant symbol, then
assign a new pebble to this node;
go to the parent
If the current pattern node is not leaf, then
let $(p_1,....,p_n)$ be the list of pebbles currently assigned to the node's sons;

/* $n \leq m$ if m is the number of sons of the current node or equivalentiy if m is the arity
of the function symbol labelling this node */
choose a pebble among $(p_1,....,p_n)$ and assign it to the node;
if the current node is the root node, then
stop

else go to the parent.

When the execution on the pattern side is terminated all the pebbles are irrelevant except those occurring at the root symbols.

It has to be stressed that the "choose" instruction is the most critical one on the pattern side.
These instructions are executed in parallel on all the patterns. Let's consider two distinct nodes $N1$ and $N2$ on the pattern sides. Let $(p_1,....,p_n)$ and $(q_1,....,q_n)$ be the lists of pebbles associated to respectively $N1$'s sons and $N2$'s sons. If $(p_1,....,p_n)$ and $(q_1,....,q_n)$ have common elements, then it can happen that Ni and $N2$ are assigned the same pebble, even if they actually denote two distinct subpatterns, so that they should get instead distinct pebbles. This kind of conflicts in pebble assignment needs to be solved when translating the pebble game in a full parallel algorithm. The patterns are the pebbles source. Pebbles moves on the term side are completely driven by pebbles moves on the patterns side:

Term side:

Leaf nodes labelled by a variable symbol, say x:
do nothing
Leaf nodes labelled by a constant symbol, say c:
whenever a pebble is assigned to a node labelled by c on the patterns side, get this same pebble
/* pebbles addition */
whenever a pebble is assigned to a node labelled by a symbol different from c on the patterns side, then
erase this same pebble
/* pebbles deletion */

Non leaf nodes, labelled by a function symbol, say f with arity m:

let $(p_1,....,p_n)$ with $n \leq m$ be the current pebbles list of this node's sons:
whenever a pebble is assigned to any node N labelled by f on the patterns side, then
let $(q_1,....,q_n)$ with $l \leq m$, be the current pebbles list of N's sons:
if $(p_1,....,p_n) = (q_1,....,q_n)$ then
get this same pebble /* pebbles addition */
erase this same pebble /* pebbles deletion */ whenever a pebble is assigned to any node N labelled by a symbol different from f on the patterns side, then
erase this same pebble

A node on the term side gets more pebbles to mean that it matches many subpatterns and eventually many patterns.
Several control issues such that the one pointed above about the "choose" instruction arise.

Let's define a parallel step as the execution unit where all nodes, i.e. all PEs, perform a pebble move. It is generally necessary that during a single parallel step all the "read" operations on pebbles happen before the "write" operations on pebbles.
Let's see an example:

Example: initial state:

$$Term : f(g(a,h(b)), f(b, f(g(a,h(b)),a)))$$

$$Patterns : f(b, x), g(a,y), f(g(z,w), v)$$

After the first parallel step:

$$Term : f(g^-(a^+, h(b^*)), f(b^*, f(g^-(a^+, h(b^*)), a^+)))$$

$$Patterns : f(b^*, x), g(a^+,y), f(g^-(z,w), v)$$

Final state, after the second parallel step:

$$Term : f^-(g^+(a, h(b)), f^*(bf^-(g^+(a, h(b)), a)))$$

$$Patterns : f^*(b, x), g^+(a, y), f^-(g(z,w), v)$$

where the *, +, - pebble occurrences denote respectively matching with the first, second and third pattern.

5. REFERENCES

[1] S.G.Akl, PARALLEL SORTING ALGORITHMS, Academic Press, New York, 1985

[2] S.Arnborg, E.Tidèn, UNIFICATION PROBLEMS WITH ONE-SIDED DISTRIBUTIVITY, In J.P. Jouannaud ed. Proceedings 1st RTA, LNCS 202, 398-406, Dijon, France, 1985

[3] J.L.Balcazar, J.Diaz, J.Gabarro, THE STRUCTURAL APPROACH TO COMPLEXITY THEORY, Preliminary draft of the monograph to appear in the Springer Verlag Serie in Theoretical Computer Science, 1988

[4] M.Bauderon, B.Courcelle, GRAPH EXPRES-

SIONS AND GRAPH REWRITING, Technical Report, Université de Bordeaux 1, 1985

[5] D.Benan av, D.Kapur, P.Narendran, COMPLEXITY OF MATCHING PROBLEMS, In J.P.Jouannaud ed. Proceedings 1st RTA, LNCS 202, Dijon, France, 1985

[6] D.Benanav, D.Kapur, P.Narendran, COMPLEXITY OF MATCHING PROBLEMS, in J. Symbolic Computation, 3, 203-216, 1987

[7] A.Boudet, J.P.Jouannaud, M.Schmidt-Schauss, UNIFICATION IN BOOLEAN RINGS AND ABELIAN GROUPS, To appear in Proceedings 3rd IEEE LICS, Edinburgh, July 1988

[8] J.Burghardt, A TREE PATTERN MATCHING ALGORITHM WITH REASONABLE SPACE REQUIREMENTS, in M.Dauchet, M.Nivat eds. Proceedings 13TH CAAP, LNCS 299, Nancy, France, 1988

[9] W. Buttner, UNIFICATION IN DATASTRUCTURES MULTISETS, J.Automated Reasoning 2,75-88, 1986

[10] C.L.Chang, R.C.Lee, SYMBOLIC LOGIC AND MECHANICAL THEOREM PROVING, Academic Press, New York, 1973

[11] C.Dwork, P.Kanellakis, J.Mitchell, ON THE SEQUENTIAL NATURE OF UNIFICATION, J. of Logic Programming, 1,1, 35-50, 1984

[12] C.Dwork, P.Kanellakis, L.Stockmeyer, PARALLEL ALGORITHMS FOR TERM MATCHING, Technical Report, M.I.T., 1986

[13] C.Dwork, P.Kanellakis, L.Stockmeyer, PARALLEL ALGORITHMS FOR TERM MATCHING, in J.Siekmann ed. Proceedings 8th CADE, LNCS 230, 416-430, Oxford, England, 1986

[14] M.Garey, D.Johnson, COMPUTERS AND INTRACTABILITY: A GUIDE TO THE THEORY OF NP-COMPLETENESS, Freeman, 1978

[15] J.A.Goguen, C.Kirchner, J.Meseguer, MODELS OF COMPUTATION FOR THE REWRITE RULE MACHINE, Technical Report R-104, CRIN, 1986

[16] C.M.Hoffman, M.J.O'Donnel, PATTERN MATCHING IN TREES, J.ACM 29, 1, 68-95, 1982

[17] J.M.Hullot, ASSOCIATIVE-COMMUTATIVE PATTERN MATCHING, in Proceedings 6th IJCAI, Tokyo, 1, 406-412, 1979

[18] ICOT WorKing Group WG.5, SEVERAL ASPECTS ON UNIFICATION, Technical Memorandum n. TM-0046, ICOT Research Center, February 1984

[19] A.Kandri-Rody, D.Kapur, P.Narendran, AN IDEAL THEORETIC APPROACH TO WORD PROBLEMS AND UNIFICATION PROBLEMS OVER FINITELY PRESENTED COMMUTATIVE ALGEBRAS, in J.P.Jouannaud ed. Proceedings 1st RTA, LNCS 202, Dijon, France, 345-364, 1985

[20] D.Kapur, P.Narendran , NP-COMPLETENESS OF THE SET UNIFICATION AND MATCHING PROBLEMS, in J.Siekmann ed. Proceedings 8th CADE, LNCS 230, 489-495, Oxford, England, 1986

[21] D.Kapur, P.Narendran, NP-COMPLETENESS OF ASSOCIATIVE-COMMUTATIVE UNIFICATION CHECK AND RELATED PROBLEMS, Submitted to The Journal of Automated Reasoning, Dec. 1986

[22] D.Kapur, P.Narendran, MATCHING, UNIFICATION AND COMPLEXITY, (A Preliminary Note), SIGSAM Bulletin, Vol. 21, No. 4, Nov. 1987

[23] D.Kapur, G.Sivakumar, H.Zhang, RRL: A REWRITE RULE LABORATORY, in J.Siekmann ed. Proceedings 8th CADE, LNCS 230, 691-692, Oxford, England, 1986

[24] R.M.Karp, R.E.Miller, A.Rosenberg, RAPID IDENTIFICATION OF REPEATED PATTERNS IN STRINGS, TREES AND ARRAYS, Proceedings 4th ACM STOC, 125-136, 1972

[25] R.M.Karp, E.Upfal, A.Wigderson, CONSTRUCTING A PERFECT MATCHING IS IN RANDOM NC, Proceedings 17th ACM STOC, 22-32, 1985

[26] C.Kirchner, METHODS AND TOOLS FOR EQUATIONAL UNIFICATION, Proceedings Colloquium on the Resolution of Equations in Algebraic Structures, Austin, Texas, May 1987

[27] D.Kozen, COMPLEXITY OF FINITELY PRESENTED ALGEBRAS, Report No. 76-294, Dept. of Computer Science, Cornell University, Ithaca, NY, cited in [14], p. 253

[28] L.Kronsjo, COMPUTATIONAL COMPLEXITY OF SEQUENTIAL AND PARALLEL ALGORITHMS, John Wiley & Sons, 1985

[29] W.Kuchlin, SOME REDUCTION STRATEGIES FOR ALGEBRAIC TERM REWRITING, Sigsam Bull., 16, 4, 13-23, Nov. 1982

[30] D.Lankford, A UNIFICATION ALGORITHM FOR ABELIAN GROUP THEORY, Technical Report MTP-1, Louisiana Techn. University, 1979

[31] D.Lankford, G.Butler, B.Brady, ABELIAN GROUP UNIFICATION ALGORITHMS FOR ELEMENTARY TERMS, General Electric Workshop on the Rewrite Rule Laboratory, April 1984

[32] J.L.Lassez, M.J.Maher, K.Marriott, UNIFICATION REVISITED, IBM T.J. Watson Research, Yorktown Heights, NY 10598, USA, Presented at Workshop on Foundations of Logic and Functional Programming, Trento, Italy, Dec. 1986

[33] P.Lincoln, G.Christian, ADVENTURES IN ASSOCIATIVE COMMUTATIVE UNIFICATION (A SUMMARY), in E.Lusk, R.Overbeek eds. Proceedings 9th CADE, LNCS 310, 358-367, Chicago, U.S.A., 1988

[34] M.Livesey, J.Siekmann, TERMINATION AND DECIDABILITY RESULTS FOR STRING UNIFICATION, Essex University, Memo CSM-12, 1975

[35] M.Livesey, J.Siekmann, UNIFICATION OF A+C-TERMS (BAGS) AND A+C+I-TERMS (SETS), Internal Report 5/76, Institut fur Informatik I, Karlsrhue University, 1976

[36] U.Martin, T.Nipkow, UNIFICATION IN BOOLEAN RINGS, in J.Siekmann ed. Proceedings 8th CADE, LNCS 230, 506-513, Oxford, England, 1986

[37] A.Martelli, U.Montanari, AN EFFICIENT UNIFICATION ALGORITHM, ACM Transactions on Programming Languages and Systems, 4, 2, 258-282, 1982

[38] U.Montanari, J.Goguen, AN ABSTRACT MACHINE FOR FAST PARALLEL MATCHING OF LINEAR PATTERNS, SRI International, 333 Ravenswood Ave., Menlo Park CA 94025, Feb. 1987

[39] J.Mzali, MATCHING WITH DISTRIBUTIVITY, in J.Siekmann ed. Proceedings 8th CADE, LNCS 230, 496-505, Oxford, England, 1986

[40] M.S.Paterson, M.N.Wegman, LINEAR UNIFICATION, J. Computer and System Sciences 16, 158-167, 1978

[41] N.Pippenger, ON SIMULTANEOUS RESOURCE BOUNDS, in Proc. 20th IEEE Symp. on Foundations of Computer Science, 307-311, 1979

[42] G.D.Plotkin, BUILDING IN EQUATIONAL THEORIES, Machine Intelligence, 7, 1972

[43] P.W.Purdom, C.A.Brown, FAST MANY-TO-ONE MATCHING ALGORITHMS, in J.P.Jouannaud ed. Proceedings 1st RTA, LNCS 202, Dijon, France, 1985

[44] M.O.Rabin, PROBABILISTIC ALGORITHMS FOR TESTING PRIMALITY, J. Number Theory 12, 128-138, 1980

[45] R.Ramesh, I.V.Ramakhrishan, OPTIMAL SPEEDUPS FOR PARALLEL PATTERN MATCHING IN TREES, in P.Lescanne ed. Proceedings 2nd RTA, LNCS 256, 274-285, Bordeaux, France, 1987

[46] P.Raufels, J.Siekmann, UNIFICATION OF IDEMPOTENT FUNCTIONS, Internal Report, Institut fur Informatik I, Karlsrhue University, 1978

[47] J.A.Robinson, A MACHINE ORIENTED LOGIC BASED ON THE RESOLUTION PRINCIPLE, J.ACM 12, 23-41, 1965

[48] M.Schmidt-Schauss, UNIFICATION IN A COMBINATION OF ABSTRACT DISJOINT EQUATIONAL THEORIES, in E.Lusk, R.Overbeek eds. Proceedings 9th CADE, LNCS 310, 379-396, Chicago, U.S.A., 1988

[49] U.Schoning, COMPLEXITY AND STRUCTURE, LNCS 211, Springer Verlag, 1985

[50] J.Siekmann, STRINGUNIFICATION, Essex University, Memo CSM-7, 1975

[51] J.Siekmann, UNIFICATION OF COMMUTATIV TERMS, Internal Report, Institut fur Informatik I, Karlsrhue University, 1976

[52] J.Siekmann, P.Szabo, A NOETHERIAN AND CONFLUENT REWRITE SYSTEM FOR IDEMPOTENT SEMIGROUP, Semigroup Forum, 25, 1982

[53] J.Siekmann, P.Szabo, A MINIMAL UNIFICATION FOR IDEMPOTENT FUNCTIONS, Internal Report, Institut fur Informatik I, Karlsrhue University, 1982

[54] M.Stickel, A COMPLETE UNIFICATION ALGORITHM FOR ASSOCIATIVE COMMUTATIVE FUNCTIONS, in Proceedings 4th IJCAI, Tbilisi, 71-76, 1975

[55] M.Stickel, A UNIFICATION ALGORITHM FOR ASSOCIATIVE COMMUTATIVE FUNCTIONS, J.ACM, 28, 3, 423-434, July 1981

[56] P.Szabo, THEORY OF FIRST ORDER UNIFICATION, Karlsrhue University, Thesis (in German), 1982

[57] R.M.Verma, I.V.Ramakhrishan, OPTIMAL TIME BOUNDS FOR PARALLEL PATTERN MATCHING, in E.Lusk, R.Overbeek eds; Proceedings 9th CADE, LNCS 310, 694-703, Chicago, U.S.A., 1988

[58] J.S.Vitter, R.A.Simons, NEW CLASSES FOR PARALLEL COMPLEXITY: A STUDY OF UNIFICATION AND OTHER COMPLETE PROBLEMS IN P, Technical Report n. CS-84-06, Department of Computer Science, Brown University Providence, October 1984

[59] T.Winkler, S.Leinwand, J.A.Goguen, SIMULATION OF CONCURRENT REWRITING, Technical Report SRI International, 1986

[60] H.Yasuura, ON THE PARALLEL COMPUTATIONAL COMPLEXITY OF UNIFICATION, Technical Report n. ER-83-01, ICOT Yajima Lab, October 1983

AVVENIMENTI

OPTICAL INFORMATION SYSTEMS 1988

(Londra, 13-15 giugno 1988)

Giancarlo Dalto, Renzo Magni
Dipartimento di Scienze dell' Informazione
Università di Milano

L'interesse per lo sviluppo delle tecnologie ottiche, la loro utilizzazione nella progettazione dei sistemi informativi e il loro mercato, cresce in misura costante anche in Europa e l'edizione 1988 dell'"Optical Information Systems" (OIS) ne rappresenta un valido esempio.
Infatti, dopo Seattle, era questa la prima occasione per verificare la consistenza dell'industria europea di fronte ai colossi americani: le maggiori industrie del "vecchio continente" non hanno mancato l'appuntamento dimostrando il buon livello raggiunto dai sistemi proposti e dalla qualità dei programmi di sperimentazione. Un dato oggettivo è rappresentato dal numero degli espositori: più di 100 contro i 45 del 1987 e i 20 del 1986.
L'aspetto più interessante del convegno riguardava l'integrazione tra acquisizione elettronica delle immagini e memorie ottiche di grande capacità (WORM): in particolare veniva sottolineato come queste applicazioni possono rivoluzionare la progettazione dei sistemi informativi e gestionali aziendali (sostituendo validamente, per esempio, i tradizionali microfilms). Ma anche il mondo dell'editoria, con l'apparizione di una nuova generazione di processori adatti a trattare dati fortemente compressi che permettono la memorizzazione su un unico supporto di immagini, dati, audio, non potrà, in un prossimo futuro, non essere sconvolto.
Nell'ambito delle applicazioni destinate ad un mercato più vasto, comprendente cioè anche quella fascia potenziale di acquirenti non specializzati, il CD-ROM, pur costituendo sempre uno strumento assai valido, dato lo stato attuale della tecnologia è stato relegato essenzialmente a scopi di tipo professionale o aziendale, mentre l'attenzione per prodotti di consumo "appetibili" è stata puntata sul CD-I, grazie alla possibilità di costruire applicazioni domestiche ed educative che vedono la trasformazione del comune televisore da passivo mezzo di intrattenimento a strumento interattivo.
Comunque sia ciascuna applicazione comincia ad avere un proprio ambito specializzato e un proprio mercato; per tale motivo e per la complessità correlata alla tecnologia ottica, il congresso è stato suddiviso in sei aree tematiche con un alto numero di interventi per traccia e, a maggiore dimostrazione dell'interesse suscitato, un ricco dibattito a conclusione di ogni relazione. Schematicamente queste sei tracce riguardavano:

- la tecnologia dei CD-I, DV-I e ICVD (Interactive Compact VideoDisk), le applicazioni e gli sviluppi anche in vista dell'approccio multimediale;

- i processi di trattamento di testi e immagini;

- l'uso della tecnologia WORM con particolare riferimento ai problemi relativi agli standards e alle interfacce a livello PC, mini e mainframe;

- l'applicazione di CD-R0M e WORM nell'ambito della

organizzazione e della distribuzione dell'informazione tecnico-scientifica e manageriale;

- il mercato dei "media" ottici con particolare riguardo ad alcuni prodotti destinati alle industrie farmaceutiche, delle costruzioni e alle imprese di assicurazione;

- gli strumenti di acquisizione delle immagini (scanner), i riconoscitori di caratteri (OCR) e le tecniche di compressione delle immagini; inoltre un beve accenno agli sviluppi futuri della tecnologia con qualche accenno alla tecnologia dei dischi "erasable-rewritable".

Per permettere una maggiore comprensione degli argomenti trattati la conferenza è stata preceduta da una serie di "workshop" che focalizzavano i vari aspetti della tecnologia dei dischi ottici dalla progettazione alle applicazioni e al mercato.
Come si diceva, erano presenti tutti i maggiori fornitori europei di sistemi di gestione e acquisizione di immagini e documenti basati su dischi WORM collegati a PC, ambienti mainframe e minicomputer che includevano anche jukeboxes; erano inoltre presenti produttori hardware e software per CD-ROM.
Schematizzando si potevano individuare, anche all'interno dell'exhibition, quattro "aree tematiche".
L'area maggiormente rappresentativa per numero di espositori e prodotti riguardava i sistemi di "Document Image Processing" dove si distingueva la British Olivetti col proprio sistema integrato "FILE NET" con la stazione di ingresso per i documenti basata su scanner ottico, il sistema di archiviazione a jukebox e la stazione di lavoro per il recupero e la stampa dei documenti. Seguiva poi la Philips presente con l'ormai noto MEGADOC e il relativo range di applicazioni. Restando nell'ambito delle grandi aziende, la Kodac presentava ufficialmente il sistema completo multiutente KIMS-3000 oltre al recente "imaging system" OD-1000 basato su PC. Tutti questi sistemi utilizzano la tecnologia WORM che di fatto sta diventando sempre più la tecnologia preferita per l'"home publishing".
Oltre alle grandi imprese era presente un agguerrito numero di piccole e medie imprese che presentavano i propri sistemi di "document imaging" a basso costo basati su PC per l'archiviazione della documentazione tecnica e amministrativa in ambito aziendale.
Seguiva poi un'area di particolare interesse per gli utenti di sistemi CAD relativa all'acquisizione e gestione di disegni tecnici, anche di grandi dimensioni. Tra le diverse aziende del settore si distinguevano la GTX e la Spectrum per i loro sistemi comprendenti "drawing scanner", software e tools per il "raster editing", facilities per la conversione di disegni in formato vettoriale utilizzabili su diversi sistemi CAD.

Una terza area riguardava i sottosistemi per l'archiviazione su dischi WORM; quest'area comprendeva una serie di nuovi drives per dischi da 5.25 e 12 pollici sia per PCs che per mainframes (IBM, VAX, APPLE, ecc.), jukeboxes e schede di interfaccia per gestione di fax.
Chiudeva l'area riguardante l'hardware, il software e i servizi di "data preparation" per CD-ROM. Accanto a prodotti già affermati come i database su CD-ROM presentati da MicroInfo e Silverplatter, si è visto anche qualche nuovo prodotto come, ad esempio, la soluzione per l'indicizzazione e la ricerca di documenti digitalizzati proposta dalla BRS Europa che migliora notevolmente le prestazioni del precedente pacchetto "BRS Search" per la ricerca "full text". La 3M invece presentava il suo disco ottico riscrivibile da 5.25 pollici con capacità di 650 Mb. Il disco, che sfrutta la tecnologia magneto-ottica, dovrebbe garantire almeno 1 milione di cancellazioni e una stabilità dei dati per almeno 30 anni.

RECENSIONI

S. Margherita, P. Terna, RASSEGNA DI STRUMENTI INFORMATICI, G.Giappidelli Editore, Torino, 1988

L'idea di raccogliere in un volume la descrizione degli strumenti informatici essenziali per un ricercatore è apprezzabile perché, se è vero che non mancano sul mercato manuali specifici di tutti gli strumenti disponibili, la ricerca di quelli più adatti può risultare dispersiva e può portare a scelte non soddisfacenti.
Lo spirito informatore del volume "Rassegna di strumenti informatici" di S. Margarita e P. Terna è dunque sostanzialmente buono, anche perché la rassegna è condotta con notevole apertura nei riguardi di strumenti come ad esempio Lisp, Prolog e muMath, di uso non abituale tra gli utenti degli strumenti informatici.
Il libro, forse eccessivamente voluminoso, è articolato in tre parti sostanzialmente indipendenti, la prima delle quali è dedicata ai linguaggi di programmazione, cioè agli strumenti informatici più generali, la seconda a strumenti più specifici, come fogli elettronici, editor e gestori di dati e la terza ad un particolare strumento realizzato da uno dei due autori (P. Terna).
Per quanto riguarda la prima parte gli autori hanno fatto la scelta di descrivere in parallelo i due linguaggi classici Fortran e Basic e tale scelta non può non suscitare qualche perplessità, se si considera il tipo di pubblico cui il volume è rivolto, anche se si deve constatare che in effetti la evoluzione dei due linguaggi non ha avvicinato molte caratteristiche.
Piuttosto ci si può rammaricare dell'assenza di un sia pure sintetico accenno all'esistenza del Pascal, almeno in riferimento alle caratteristiche più significative di questo linguaggio, visto che nella premessa gli autori si propongono di "guidare il lettore verso un uso rigoroso e consapevole degli strumenti informatici".
Già si è espresso invece apprezzamento per la scelta di inserire tra i linguaggi di programmazione anche il Lisp ed il Prolog, che non sono certo linguaggi popolari tra i ricercatori utenti di Informatica. Sarebbe stato auspicabile tuttavia, almeno nell'introduzione al Lisp, un sia pur breve inquadramento sulle finalità del linguaggio, la cui descrizione rischia altrimenti di risultare del tutto gratuita e incomprensibile.

La seconda parte del volume sembra la più coerente con il titolo in quanto viene illustrato un certo numero di strumenti informatici che possono essere di grande utilità per un ricercatore che dispone di un personal computer. La scelta degli strumenti è del tutto ragionevole e oltre a strumenti ormai tradizionali come i fogli elettronici (Multiplan e Lotus 1-2-3) il sistema di gestione di dati BASE III Plus, il classico pacchetto statistico SPSS/PC+ e i text editor (Personal Editor II) comprende anche uno strumento molto meno noto, come muMATH, di interesse già oggi rilevante e probabilmente destinato ad una maggiore diffusione nel futuro.
Trattandosi della descrizione di strumenti, l'espressione diventa necessariamente dettagliata e tuttavia è svolta con una buona sistematicità con frequenti riferimenti a funzioni simili svolte da strumenti diversi. La rassegna si conclude con un capitolo dedicato alle possibilità di comunicazione tra ambienti software diversi, problema questo che spesso pone in serie difficoltà gli utenti non espertissimi.
La terza parte del volume risulta la più discutibile perché dedicata alla descrizione di uno strumento, di interesse meno generale di quelli illustrati nelle parti precedenti, con un livello di dettaglio eccessivo.

a cura di Mario Italiani

PROOF THEORY AND LOGICAL COMPLEXITY, Jean-Yves Girard, Bibliopolis, Napoli, 1987, pp. 503

" Studies in Proof Theory " è una collana della casa editrice napoletana Bibliopolis che pubblica testi ri-guardanti quella parte della Logica Matematica che considera come oggetto di studio le dimostrazioni formali e gli strumenti matematici ad esse correlati. In questa collana è apparso il volume di cui ci occupiamo; l'autore, Jean-Yves Girard, ricercatore in logica matematica di fama internazionale, è membro del CNRS (l'ente di stato per la ricerca in Francia) e dell'Université VII di Parigi.
" Proof Theory and Logical Complexity " è diviso in due volumi, ma al momento in cui scriviamo è apparso solo il primo volume. Nelle intenzioni dell'autore quest'opera dovrebbe costituire una introduzione alla teoria della dimostrazione; tuttavia, a nostro parere, si tratta di una introduzione in un certo senso avanzata visto che tra gli argomenti trattati (in special modo nel secondo volume a venire) troviamo i frutti di ricerche piuttosto recenti. Per di più - come lo stesso autore ammette nella presentazione - gli aspetti più comuni della teoria della dimostrazione sono in un certo senso trascurati e il lettore viene rimandato per questi ad altri testi, come il testo di G. Takeuti (" Proof Theory "), quello di K. Schütte (" Proof Theory ") - Springer Verlag, Berlin 1977) o quello classico di S. C. Kleene (" Introduction to metamathematics " - North Holland, Amsterdam, 1952).

Ancora nella presentazione che Girard fa del suo testo troviamo alcune osservazioni che è opportuno tener pre-

sente se vogliamo considerare l'opera da un corretto punto di vista.
E' opportuno intanto ricordare che cosa l'autore intende per teoria della dimostrazione. Per prima cosa Girard rifiuta contrapposizione tra sintassi e semantica come viene usualmente presentata nella logica: della sintassi si occuperebbe in particolare la teoria della dimostrazione: cosicché le dimostrazioni formali sarebbero viste come oggetti sintattici particolarmente strutturati; mentre la semantica sarebbe dominio della teoria dei modelli. L'autore preferisce pensare che la teoria della dimostrazione si occupi più in generale del rapporto tra gli oggetti matematici (che in generale sono infiniti) e le loro rappresentazioni (necessariamente finite).
Inoltre viene esposto il principio generale secondo cui è organizzato il libro. Gli argomenti sono ordinati e presentati seguendo la loro complessità: così si inizia con una introduzione nella quale, tra l'altro, sono presentati i risultati di base della teoria della dimostrazione; quindi nella prima parte si espone la logica usuale al primo e al secondo ordine. Nella seconda parte si passa alla logica infinitaria, e così via proseguendo nel secondo volume.

Così il primo volume dell'opera (cioè quello che stiamo esaminando) è suddiviso in due parti precedute da un capitolo a carattere introduttivo. In questo capitolo iniziale si parla soprattutto del programma di Hilbert: sinteticamente lo scopo del programma suddetto era quello di ridurre, e in un certo senso giustificare, la matematica ideale (quella che fa uso di strumenti astratti) alla matematica reale (quella concreta, verificabile con strumenti finiti o quasi); se ne illustra il fallimento e le sue conseguenze. A tal fine vengono illustrati gli strumenti essenziali della ricorsività, particolare settore della logica che tratta della calcolabilità e sviluppatosi proprio in relazione allo sviluppo programma di Hilbert; sono presentati infine i teoremi di incompletezza di Gödel. Il capitolo si conclude con un'appendice nella quale si introducono intuizionismo e costruttivismo.

Il testo però entra nel vivo con il secondo capitolo - il primo della prima parte dedicata alla logica usuale al primo e al second'ordine.
Per primo viene presentato il calcolo dei sequenti di Gentzen sia per la logica classica che per quella intuizionista. Viene illustrato il Teorema di Eliminazione dei Tagli (noto come Haupsatz, dimostrato dallo stesso Gentzen nel 1934) e vengono presentate le sue principali conseguenze: Proprietà della Sottoformula, teorema di Herbrand,... Nello stesso capitolo si parla del sistema di deduzione naturale per la logica intuizionista, e si dimostra il teorema di Church-Rosser e il teorema di Normalizzazione Forte di Prawitz.

Il capitolo successivo è dedicato al problema dell'eliminazione dei tagli in calcoli diversi (e in particolare per la logica al second'ordine). Per questo viene presentata una tecnica che fa uso di modelli logici a tre valori di verità.
Nel quarto capitolo sono presentati alcuni risultati che seguono dall'Haupsatz; il teorema di Interpolazione per cominciare; quindi lo Schema di Riflessione per l'aritmetica di Peano (PA). Infine vengono illustrati alcuni risultati riguardanti il problema della consistenza.
In appendice a questo capitolo troviamo un'interessante applicazione dello Haupsatz ad un caso concreto, cioè il teorema di Van der Waerden: si mostra che normalizzando la dimostrazione formale di Furstenberg e Weiss (lievemente modificata) si riottiene in sostanza la dimostrazione originale di Van der Waerden. Sempre nell'appendice al capitolo troviamo discussa la Normalizzazione per l'aritmetica di Heyting (cioè l'aritmetica trattata con la logica intuizionista) e per la logica intuizionista al second'ordine; è opportuno notare che viene qui utilizzata la tecnica del candidato di riducibilità, strumento fondamentale per la trattazione del second'ordine.
Sulla seconda parte, dato il suo carattere piuttosto tecnico, ci limiteremo a riferire che consiste di tre capitoli dedicati alla logica infinitaria (nella quale vengono trattate dimostrazioni e formule che non sono in generale oggetti finiti) e agli strumenti necessari a questa.

In tutto il libro l'autore dedica particolare attenzione alla complessità degli oggetti e delle dimostrazioni che sta trattando, fornendo - ogni volta è possibile - una valutazione di tale complessità. Inoltre nel testo - oltre ad una parte matematica di indubbio valore - troviamo spesso stimolanti esposizioni e discussioni di carattere storico e metodologico.

Restano da rilevare un paio di pecche alla presentazione editoriale del libro: innanzitutto l'indice analitico è troppo povero per un testo di questa ampiezza; in secondo luogo la veste grafica è tale da rendere difficile una veloce individuazione ad esempio di un teorema o di una definizione. Se si pensa che un'opera di questo genere è destinata ad essere uno strumento di lavoro, è spiacevole constatare che aspetti marginali possano renderne difficile la consultazione.

a cura di Ugo Solitro

EDITO DA CALEIDOGRAF S.R.L.
VIA L. DA VINCI N.4/6
OSNAGO (CO)

REALIZZATO DALLA COMPUTER ARE S.R.L. SEZ. D.T.P.
CON MACINTOSH™ DELLA APPLE COMPUTER